ПРОФЕССИОНАЛЬНАЯ РАЗРАБОТКА

В.А. Ажеронок
А.В. Островерх
М.Г. Радченко
Е.Ю. Хрусталева

РАЗРАБОТКА ИНТЕРФЕЙСА ПРИКЛАДНЫХ РЕШЕНИЙ НА ПЛАТФОРМЕ «1С:ПРЕДПРИЯТИЕ 8»

1С ПАБЛИШИНГ

В.А. Ажеронок, А.В. Островерх, М.Г. Радченко, Е. Ю. Хрусталева

Разработка интерфейса прикладных решений на платформе "1С:Предприятие 8"

Электронная книга в формате pdf; ISBN 978-5-9677-2815-0.

Электронный аналог печатного издания
«Разработка интерфейса прикладных решений на платформе 1С:Предприятие 8»
(ISBN 978-5-9677-2814-3, М.: ООО «1С-Паблишинг», 2018;
артикул печатной книги по прайс-листу фирмы «1С»: 4601546136947;
по вопросам приобретения печатных изданий издательства «1С-Паблишинг» обращайтесь к партнеру «1С», обслуживающему вашу организацию, или к другим партнерам фирмы «1С», в магазины "1С Интерес", а также в книжные и интернет-магазины).

Эта книга является обновленным и дополненным изданием книги «Разработка управляемого интерфейса».
Книга адресована специалистам, имеющим опыт разработки на платформе «1С:Предприятие 8.3». Также она будет интересна и полезна всем программистам, желающим познакомиться с тем, как создаются прикладные решения, работающие в интерфейсе «Такси».
Книга состоит из пяти частей, посвященных различным областям разработки интерфейса: формированию командного интерфейса, разработке форм в конфигураторе, программированию форм, оптимизации взаимодействия между клиентской и серверной частью приложения, адаптации форм для работы в мобильном клиенте.
Большое количество иллюстраций и практических примеров помогут читателю быстрее освоить новые технологии разработки.

Дополнительные материалы

Демонстрационные конфигурации, используемые в книге, опубликованы на портале 1С:ИТС. Вы можете использовать их для практического знакомства с примерами и для доработки в целях изучения новых возможностей платформы.

Все конфигурации созданы на версии платформы 8.3.12.1412.

Скачайте материалы и учебную версию платформы на странице http://its.1c.ru/book_demo/, раскройте архив и следуйте инструкциям по установке.

Оглавление

Введение

Концепция интерфейса, используемая в платформе «1С:Предприятие 8.3», позволяет прикладным решениям работать не только внутри локальных сетей, но и через Интернет, используя низкоскоростные каналы связи.

В основе этой концепции лежат две важные особенности. Во-первых, платформа самостоятельно формирует пользовательский интерфейс на основе статического, декларативного описания, создаваемого разработчиком. Грубо говоря, разработчик описывает некоторые правила, а платформа, следуя этим правилам, самостоятельно размещает элементы в формах и окнах. При этом разработчик, если это необходимо, имеет возможность в процессе работы прикладного решения изменять некоторые части интерфейса и создавать собственные алгоритмы обработки данных, представленных в интерфейсе. Вторая особенность заключается в том, что разработчику в явном виде необходимо программировать как клиентскую, так и серверную части своего приложения.

Для специалистов, имеющих опыт разработки в предыдущих версиях платформы «1С:Предприятие», обе эти особенности являются новыми. Для программистов, не знакомых с системой «1С:Предприятие», новыми и непривычными могут являться и другие подходы, принятые при разработке прикладных решений «1С:Предприятия».

По этой причине данная книга описывает не только технические, но и многие концептуальные вопросы, необходимые для конструирования и программирования пользовательского интерфейса.

Поскольку книга посвящена технической стороне разработки интерфейса, вопросы дизайна и юзабилити в ней не рассматриваются.

Однако интерфейсы должны быть красивые и удобные для пользователя, рекомендуем вам ознакомиться с «Руководством по стилю для типовых конфигураций на платформе 1С:Предприятие 8», которое опубликовано на портале ИТС (http://its.1c.ru/app-design).

Книга состоит из пяти частей. Каждая из них посвящена определенной области разработки.

В первой части книги рассказывается о принципах, на основе которых платформа формирует командный интерфейс, и о том, как разработчик может влиять на его формирование.

Во второй части рассматриваются различные возможности статического создания форм в конфигураторе – средстве разработки, входящем в состав платформы.

В третьей части рассказывается о возможностях использования встроенного языка для решения различных интерфейсных задач.

В четвертой части книги рассматриваются основные принципы и подходы, позволяющие формам работать быстро и производительно.

В пятой части рассказывается о том, как адаптировать интерфейс прикладного решения для работы в мобильном клиенте. Мобильные устройства имеют маленькие экраны, поэтому платформе нужны дополнительные «подсказки» от разработчика, чтобы она могла «подстроить» внешний вид формы под маленький экран.

Помимо большого количества иллюстраций и практических примеров книга содержит значительное число демонстрационных конфигураций. Они опубликованы на портале 1С:ИТС. Вы можете скачать их по адресу http://its.1c.ru/download/book_demo/.

Каждая из конфигураций – это отдельный пример, рассматриваемый в книге. Таким образом, можно самостоятельно воспроизвести или доработать любой пример из книги, используя имеющиеся готовые решения.

Все демонстрационные конфигурации созданы на версии платформы 8.3.12.1412. При описании мобильного клиента использовалась мобильная платформа «1С:Предприятие» версии 8.3.12.64.

ЧАСТЬ 1

Конструирование интерфейса

Оглавление

Глава 1.1. Пользователь, интерфейс, команда

В книге часто будут встречаться термины «пользователь», «пользовательский интерфейс», «команда» и «командный интерфейс». Давайте определим, что означают эти термины и как они связаны.

Пользователь – человек, работающий с прикладным решением в режиме 1С:Предприятие. Автоматизированная система создает рабочую среду, взаимодействуя с которой, пользователь решает стоящие перед ним задачи. Для обеспечения удобной и безопасной работы создаются пользовательские интерфейсы.

Пользовательский интерфейс – интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Пользовательский интерфейс компьютерной программы часто понимают только как ее внешний вид. В действительности пользовательский интерфейс объединяет в себе все элементы и компоненты программы, которые влияют на взаимодействие пользователя с программным обеспечением. Основными элементами пользовательского интерфейса являются:

Определение взято из свободной интернет-энциклопедии «Википедия»: http://ru.wikipedia.org/wiki/Интерфейс_пользователя

- набор задач пользователя, которые он решает при помощи программы;
- элементы управления программой;
- навигация между разделами программы;
- дизайн экранов программы;
- отображаемая информация и форматы отображения;
- устройства и технологии ввода данных;
- диалоги, взаимодействие и транзакции между пользователем и компьютером;
- обратная связь с пользователем.

Составной частью пользовательского интерфейса является командный интерфейс, обеспечивающий доступ пользователей к функциональности программы.

ПРИМЕЧАНИЕ

В дальнейшем мы будем оперировать в основном понятием «командный интерфейс». Однако принципы формирования командного интерфейса во многом аналогичны принципам формирования прочих составных частей пользовательского интерфейса, в частности форм.

В общем случае *команда* – это указание некоему интерфейсу (командный интерфейс пользователя), побуждающее исполнителя (компьютерная программа) выполнить действия для решения задачи.

Определение взято из свободной интернет-энциклопедии «Википедия»: http://ru.wikipedia.org/wiki/Команда_(программирование)

При этом команда:

- может быть отдана только пользователем, имеющим на это право;
- не включает в себя подробные разъяснения, как будет происходить исполнение, а предполагает, что исполнитель (программа) знает, что и как необходимо исполнить.

Командный интерфейс – совокупность команд, доступных пользователю, при помощи которых он отдает приказание системе на выполнение действий.

Применительно к компьютерной программе совокупность команд определяет, какие действия она может выполнить, а командный интерфейс определяет, какие команды может отдать конкретный пользователь и как он получит доступ к этим командам.

Набор команд интерактивной компьютерной программы обычно включает в себя несколько категорий команд:

- команды, которые позволяют перемещаться между функциональными блоками программы, – например, функции настройки программы, функции просмотра данных, функции модификации данных, функции импорта/экспорта данных;

- команды, которые позволяют вызывать формы и перемещаться между ними, – например, вызов формы просмотра перечня товаров, вызов формы ввода нового документа;
- команды, которые позволяют выполнять те или иные действия с обрабатываемыми данными, – например, изменение элемента справочника, построение отчета.

В связи с тем, что различные пользователи выполняют различные задачи, права на использование обрабатываемой системой информации и состав доступных команд для них будут различаться.

Для разграничения прав и состава команд в компьютерных программах обычно реализуется система учетных записей. Каждая *учетная запись* (возможно, защищенная паролем) определяет совокупность ресурсов, доступных пользователю. На основании учетной записи осуществляются идентификация, аутентификация пользователя и предоставление разрешения на работу с ресурсами.

Таким образом, при разработке командного интерфейса необходимо решить две основные задачи:

- из множества команд, исполняемых программой, выделить подмножество команд, доступных конкретному пользователю;
- предоставить возможность пользователю вызывать доступные команды.

Глава 1.2. Прикладное решение глазами пользователя

В данной главе мы рассмотрим структуру пользовательского интерфейса прикладного решения под названием «Такси». Он разработан в соответствии с современными тенденциями построения интерфейсов и отвечает требованиям работы с веб-приложениями. Интерфейс «Такси» является «дружественным» и комфортным в использовании за счет удобной и быстрой навигации по прикладному решению, настраиваемого пользователем рабочего пространства и многого другого.

Первый взгляд

Для знакомства с пользовательским интерфейсом запустим демонстрационную базу «Глава 1.2. Прикладное решение глазами пользователя».

При запуске «1С:Предприятия» открывается *основное окно* приложения. В *рабочей области* основного окна приложения мы видим *начальную страницу*, с которой начинается работа пользователя с прикладным решением (рис. 1.1).

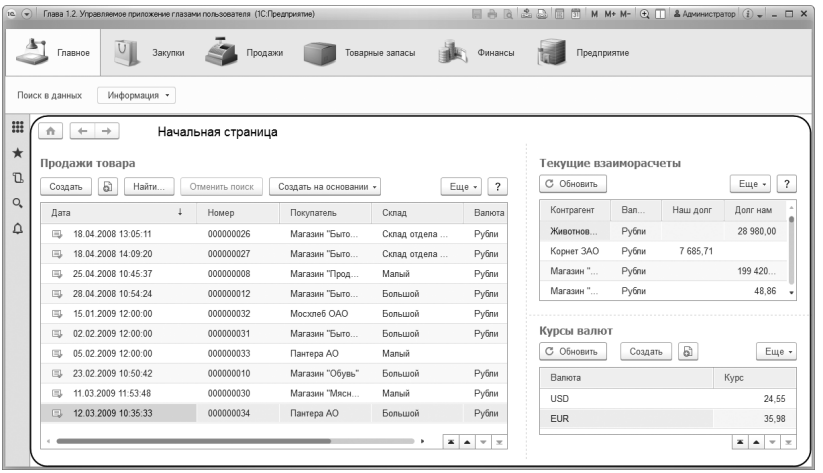


Рис. 1.1. Начальная страница прикладного решения

Задача начальной страницы – быстрое включение пользователя в работу. На начальной странице находятся формы, которые наиболее часто используются в данном прикладном решении. Например, список продаж товаров, расчеты с контрагентами, курсы валют и т. д. Причем каждый пользователь, в зависимости от своей роли, видит те формы, которые соответствуют специфике его работы.

Состав форм на начальной странице настраивается при разработке конфигурации (в т. ч. программно). Однако пользователь, исходя из своих предпочтений, также может изменить состав форм на начальной странице из списка доступных форм, предназначенных для этого разработчиком.

Более подробно о пользовательской настройке начальной страницы рассказывается в главе 2.10 «Начальная страница» на стр. 210.

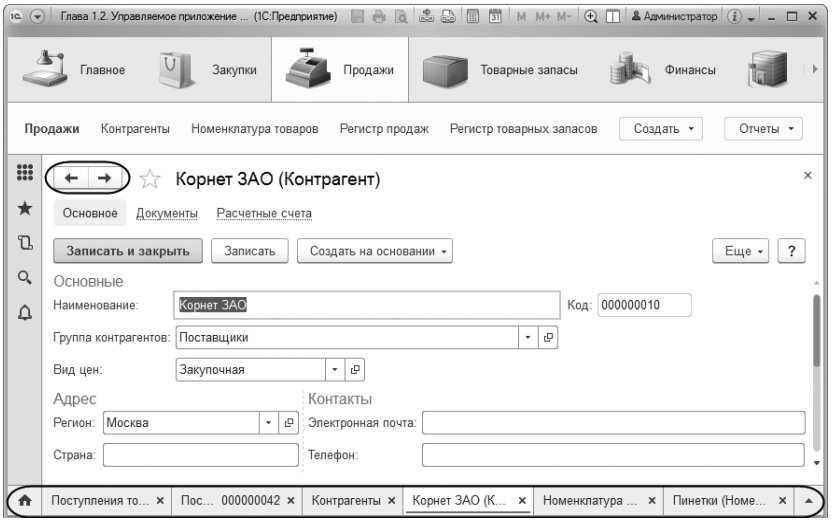


Рис. 1.2. Команды перехода между открытыми окнами и панель открытых

В рабочей области основного окна приложения открываются различные окна клиентского приложения для работы с данными информационной базы, формирования отчетов и т. п. Поскольку интерфейс «Такси» однооконный, то в каждый конкретный момент

времени в рабочей области обычно отображается какое-то одно окно клиентского приложения. Для перехода между открытыми окнами используются команды формы Вперед/Назад, а также Панель открытых (рис. 1.2).

Основное окно приложения

Для доступа к функциональности приложения в основном окне могут располагаться различные панели, с помощью которых пользователь может эффективно осуществлять навигацию по приложению, т. е. быстро находить нужные разделы и команды.

Показ этих панелей может быть включен/отключен как при разработке, так и в пользовательском режиме прикладного решения. При этом и разработчик, и пользователь могут настраивать расположение этих панелей в основном окне так, как им удобно.

Состав панелей интерфейса

Основное окно приложения содержит следующие элементы:

- область системных команд;
- панель разделов;
- панель функций текущего раздела;
- панель инструментов;
- панель избранного;
- панель истории;
- панель открытых;
- рабочая область;
- информационная панель.

Но не все из этих панелей видны при стандартной настройке интерфейса. Рассмотрим сначала стандартный вид основного окна приложения (рис. 1.3).

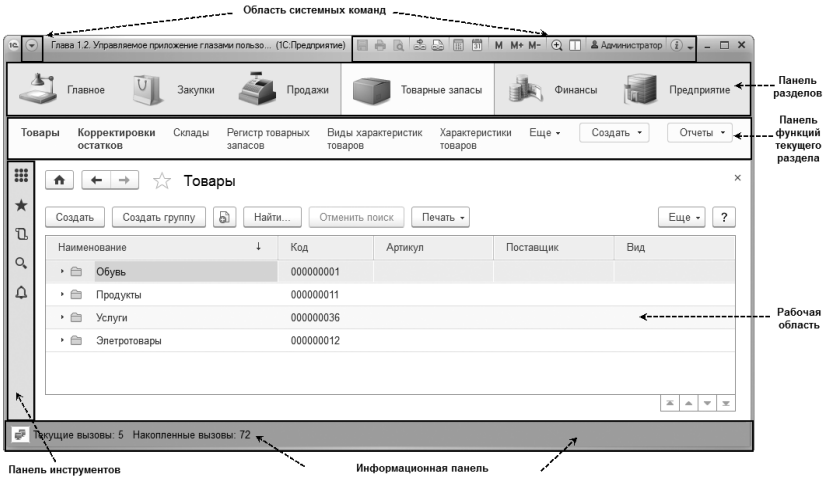


Рис. 1.3. Стандартный вид основного окна приложения

В самом верху окна находится *область системных команд* (см. рис. 1.3). В этой области выводится заголовок приложения и располагаются системные команды, состав которых не зависит от конкретного прикладного решения. Системные команды обеспечивают выполнение общих действий по управлению прикладным решением. Например, обеспечивают доступ к главному меню прикладного решения, расположенному в левой части области системных команд. Справа от заголовка приложения находятся различные вспомогательные команды, например: команды печати, получения навигационной ссылки и перехода по ней, команда быстрого масштабирования форм и т. д.

Если в информационной базе ведется список пользователей, то в области системных команд, рядом с информацией о программе, отображается гиперссылка с именем пользователя (см. рис. 1.3), при нажатии на которую появляется диалог с информацией о текущем пользователе. С помощью этого диалога пользователь также может завершить работу с клиентским приложением, при этом одновременно выполняется отмена аутентификации OpenID (если она использовалась).

Отображение остальных панелей интерфейса клиентского приложения может настраиваться как в конфигураторе (в т. ч. программно) в целом для прикладного решения, так и отдельным пользователем «под себя» в режиме 1С:Предприятие.

Подробнее о настройке видимости и отображения панелей основного окна рассказывается в разделе «Настройка панелей интерфейса» на стр. 21.

Под областью системных команд расположена *панель разделов* (см. рис. 1.3). Разделы прикладного решения делят всю функциональность приложения на части, каждая из которых содержит набор команд, объединенных прикладным смыслом. Разделы прикладного решения соответствуют подсистемам верхнего уровня иерархии, созданным в конфигурации. Если подсистем в конфигурации нет, то панель разделов не будет отображаться в основном окне приложения.

Для повышения наглядности каждой подсистеме можно установить понятную картинку. Причем и в конфигураторе, и в режиме 1С:Предприятие при настройке панели разделов можно установить расположение текста под картинкой или справа от нее.

Первым разделом всегда является основной раздел, который называется Главное. Разработчик помещает в раздел Главное команды, которые всегда должны быть «под рукой» у конкретного пользователя в зависимости от его роли.

Более подробно о пользовательских настройках панели разделов рассказывается в главе 1.8 «Пользовательская настройка интерфейса» в разделе «Настройка панели разделов» на стр. 128.

Состав и порядок остальных разделов стандартно настраиваются разработчиком в конфигураторе, но пользователь в режиме 1С:Предприятие может изменить их по своему желанию.

Если все разделы не помещаются на панели, то появляется кнопка горизонтальной прокрутки панели разделов. При повторном выборе раздела открывается меню функций, в котором собраны все команды текущего раздела.

Под панелью разделов расположена *панель функций текущего раздела* (см. рис. 1.3). В начале панели обычно расположены команды, позволяющие открыть формы различных списков. Например, выбрав раздел Товарные запасы и выполнив команду Товары, мы увидим список товаров в рабочей области окна.

Важные команды находятся в самом начале и выделены жирным шрифтом. Если все команды не помещаются в панели функций текущего раздела – на панели отображается подменю Еще, которое содержит оставшиеся команды.

Затем в подменю Создать, Отчеты и Сервис сгруппированы команды, позволяющие выполнять различные действия: создавать новые элементы данных, формировать различные отчеты или выполнять какие-то обработки. После них могут находиться произвольные группы команд, созданные разработчиком. Если команды в какой-то группе команд отсутствуют, то эта группа не отображается на панели (как, например, группа Сервис на рис. 1.3).

Вертикально, слева от рабочей области, в основном окне приложения отображается *панель инструментов* (см. рис. 1.3). Эта панель предназначена для быстрого доступа к следующим элементам интерфейса (рис. 1.4):

- Меню функций;
- Избранное;
- История;
- Поиск;
- Центр оповещений.

Рис. 1.4. Панель инструментов

Меню функций содержит все команды текущего раздела (рис. 1.5).

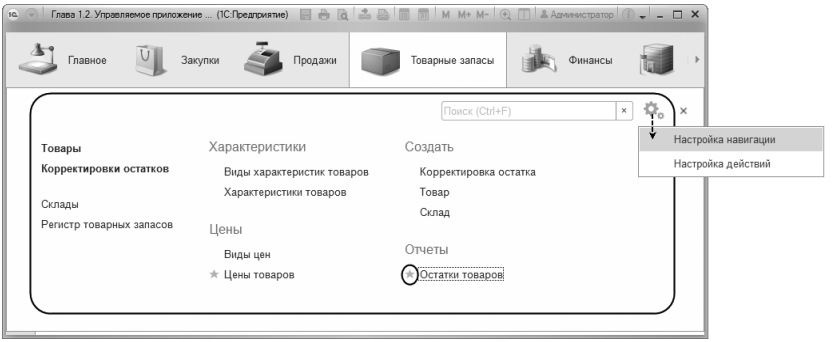


Рис. 1.5. Меню функций

Подведя курсор к часто используемой команде, пользователь может нажать на звездочку слева от нее и добавить эту команду в избранное. Повторное нажатие на звездочку удаляет команду из избранного.

В правом верхнем углу меню функций находятся поле ввода для поиска и меню настроек (см. рис. 1.5).

После ввода символов в поле ввода для поиска во всех разделах прикладного решения выполняется поиск функций, в представлении которых содержатся искомые символы (рис. 1.6).

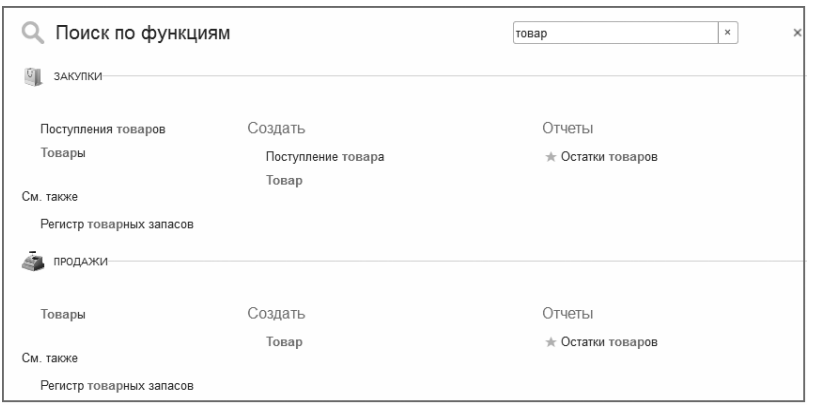


Рис. 1.6. Поиск по функциям

Из панели инструментов пользователь может открыть *избранное* (на рис. 1.7 слева), где содержится список навигационных ссылок на различные элементы прикладного решения, отмеченные пользователем для частого использования. В избранное можно добавить команды из меню функций, из истории и непосредственно из формы.

Рис. 1.7. Форма избранного и панель избранного

В избранном пользователь может выделить особенно важные элементы, которые он использует чаще всего. Для этого нужно нажать на пиктограмму кнопки, расположенной слева от элемента. Такой элемент будет выделен полужирным шрифтом и при последующем открытии избранного всегда будет расположен в верхней части списка. Можно также переименовать элемент или удалить его из избранного.

Чтобы быстрее найти нужный элемент в избранном, можно ввести искомые символы в поле ввода для поиска в правом верхнем углу формы.

Избранные ссылки могут быть также показаны в *панели избранного* (на рис. 1.7 справа), если показ этой панели включен в интерфейсе приложения. В этом случае возможности поиска, переименования, удаления и отметки важного элемента избранного становятся недоступны.

Из панели избранного можно открыть избранное, нажав на специальную пиктограмму в правом верхнем углу панели.

Из панели инструментов пользователь может открыть *историю* (на рис. 1.8 слева), в которой содержатся навигационные ссылки

на недавно открытые, созданные или отредактированные объекты информационной базы. Эти ссылки упорядочены по времени использования и сгруппированы по датам.

Рис. 1.8. Форма истории и панель истории

Таким образом, историю можно использовать для быстрого доступа к различным документам, элементам справочников и т. п., с которыми пользователь недавно работал.

Чтобы быстрее найти нужный элемент в истории, можно ввести искомые символы в поле ввода для поиска в правом верхнем углу формы.

История может быть также показана в *панели истории* (на рис. 1.8 справа), если показ этой панели включен в интерфейсе приложения. В этом случае группировка по датам и отображение времени использования объекта прикладного решения становятся недоступны.

Из панели истории можно открыть историю, нажав на специальную пиктограмму в правом верхнем углу панели.

Из панели инструментов пользователь может открыть *поиск* (если полнотекстовый поиск включен в приложении). Стандартно вызывается системная форма полнотекстового поиска (рис. 1.9).

В правой части формы отображаются последние запросы, введенные данным пользователем. Для быстрого получения результата поиска достаточно нажать гиперссылку с нужным текстом.

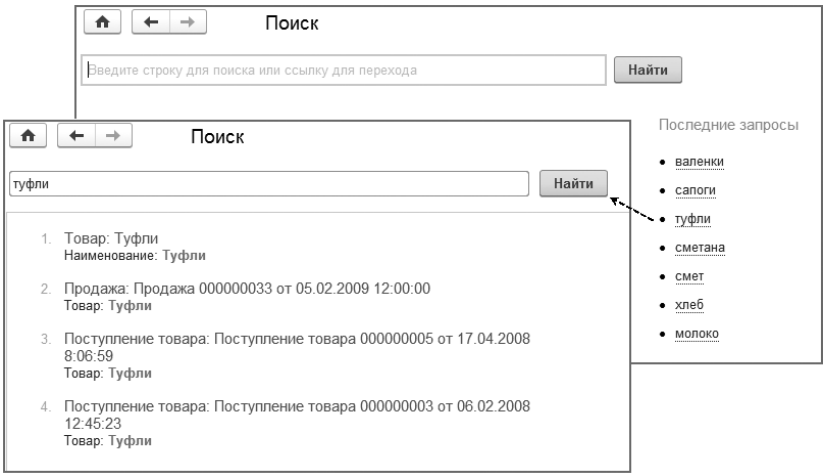


Рис. 1.9. Форма полнотекстового поиска

С помощью специального языка выражений для поиска можно выполнять полнотекстовый поиск по началу слов, «неточный» поиск, поиск по синонимам и т. д.

Из панели инструментов пользователь может открыть *центр оповещений*, который предназначен для работы с оповещениями пользователя.

Стандартно последние три оповещения показываются на несколько секунд в правом нижнем углу рабочей области основного окна, после чего пропадают. Но важные сообщения пользователю запоминаются в центре оповещений «1С:Предприятия».

Если у пользователя появились новые важные оповещения, то в панели инструментов рядом с иконкой появляется оранжевая точка. При нажатии на иконку открывается окно центра оповещений, в котором отображаются важные оповещения, на которые пользователь еще не отреагировал – не закрыл или не выполнил связанное с оповещением действие.

Самые последние оповещения всегда находятся вверху списка. Таким образом, даже если пользователь на какое-то время отходил от компьютера, он все равно сможет их увидеть (рис. 1.10).

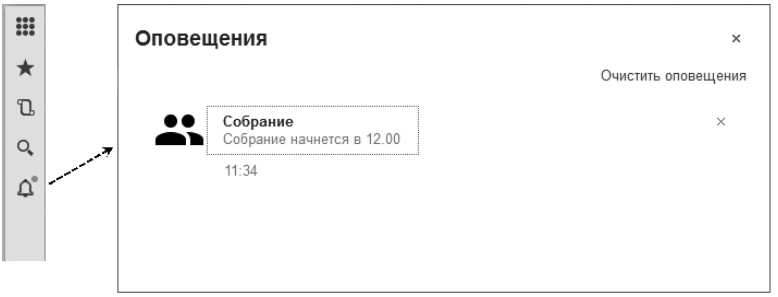


Рис. 1.10. Центр оповещений

Оповещение можно удалить из списка кнопкой очистки. Или же удалить сразу все оповещения кнопкой Очистить оповещения. При этом действия, связанные с оповещениями, выполнены не будут.

Удобным инструментом для переключения между окнами, открытыми в текущем сеансе работы с прикладным решением, является *панель открытых*. Стандартно она не видна в интерфейсе приложения. Можно настроить ее отображение (см. раздел «Настройка панелей интерфейса» на стр. 23) и поместить горизонтально внизу основного окна. В этом случае она будет выглядеть компактно, в одну строку (рис. 1.11).

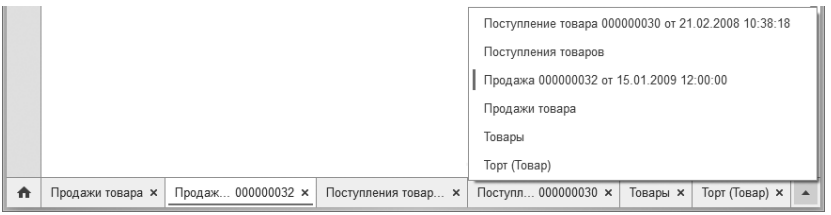


Рис. 1.11. Панель открытых

Если заголовки окон на закладках не помещаются в панели открытых, они обрезаются, заголовок начальной страницы показывается в виде иконки, а в правой части панели появляется кнопка для вызова списка всех окон. В этом списке в алфавитном порядке располагаются полные заголовки всех открытых окон. Активное окно выделяется зеленым маркером.

Когда панель открытых расположена вертикально, текст в закладках выводится без ограничений по количеству строк. Если закладки перестают помещаться по высоте, в панели появляется вертикальная полоса прокрутки. Закладка начальной страницы не прокручивается и находится в самом верху панели.

Пользователь может изменять порядок следования друг за другом открытых окон путем перетаскивания закладок в панели открытых, а также закрывать формы щелчком средней кнопки мыши (колеса) на элементе панели открытых.

В подвале основного окна приложения (см. рис. 1.3) отображается информационная панель, которая показывает, включен ли режим имитации задержек при вызове сервера и показатели производительности системы. Они нужны для анализа и оптимизации клиент-серверного взаимодействия, о котором речь пойдет в четвертой части. Отображение показателей производительности можно включить/отключить с помощью настройки параметров приложения как в конфигураторе, так и в режиме 1С:Предприятие.

Настройка панелей интерфейса

При разработке прикладного решения в конфигураторе разработчик может настроить стандартное расположение панелей интерфейса, которые будут отображаться для всех пользователей в основном окне приложения. Для этого нужно выполнить команду Открыть интерфейс клиентского приложения из контекстного меню корня конфигурации.

Например, если разработчик хочет, чтобы внизу основного окна прикладного решения отображалась панель открытых, ему нужно в окне настройки перенести Панель открытых из списка всех доступных панелей интерфейса (справа) в левую часть окна, в группу Низ (см. рис. 1.12 вверху). В этом случае основное окно приложения примет следующий вид (рис. 1.12 внизу).

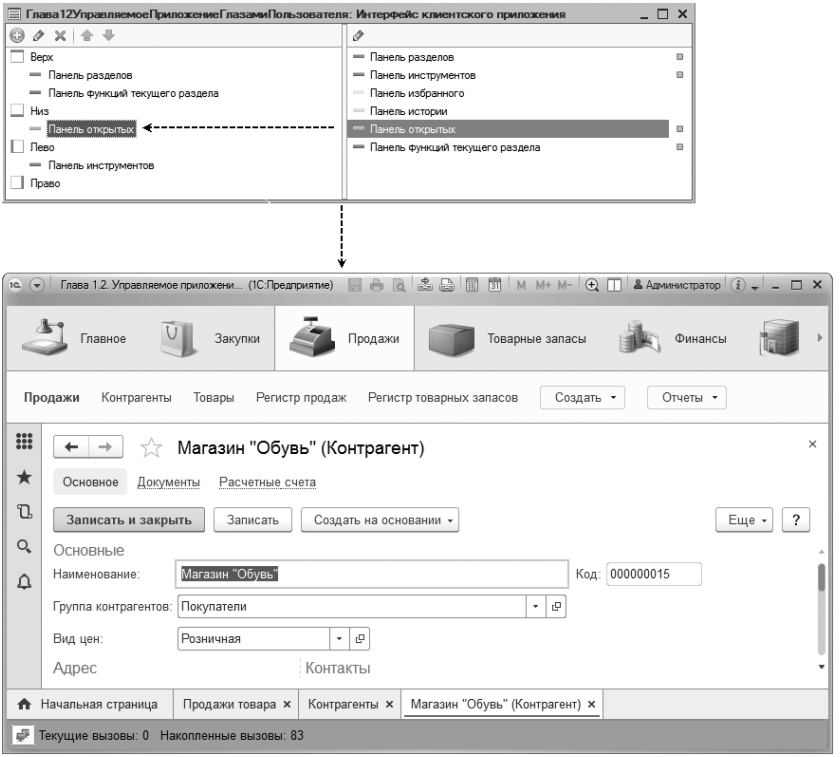


Рис. 1.12. Настройка панелей интерфейса в конфигураторе

Однако если конкретного пользователя не устраивает то, что для него настроил разработчик, он может изменить настройку панелей интерфейса лично для себя в режиме 1С:Предприятие в диалоге, вызываемом по команде главного меню Вид – Настройка панелей.

Например, в редакторе панелей пользователь может перетащить панель инструментов прямо на панель разделов, тем самым объединив обе панели в одну строку. Скрыть панель функций текущего раздела, перетащив ее в нижний серый прямоугольник (где находятся все невидимые панели), а панель избранного перетащить из нижнего прямоугольника в правый край окна (см. рис. 1.13 вверху). В этом случае основное окно приложения примет следующий вид (рис. 1.13 внизу).

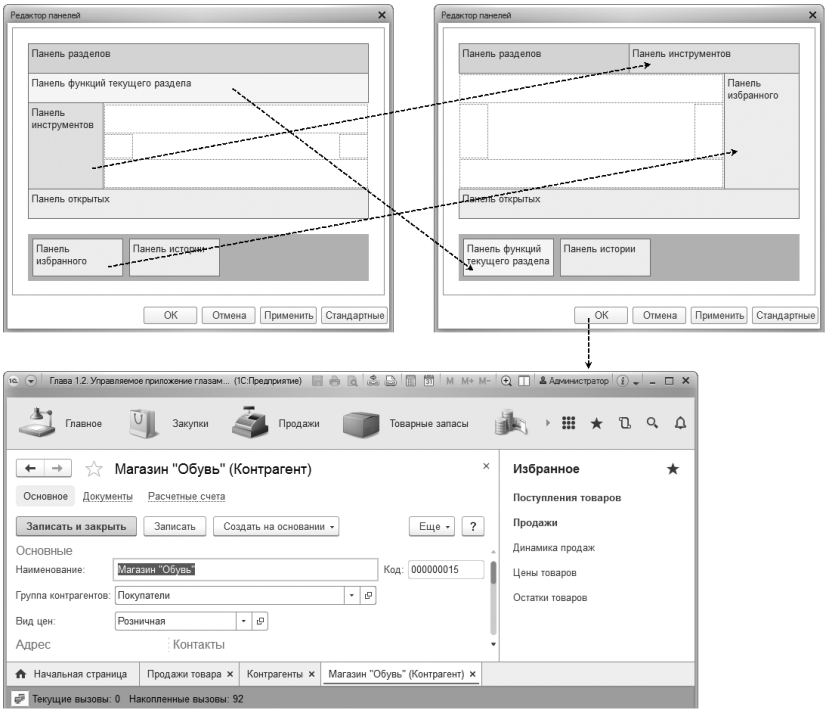


Рис. 1.13. Настройка панелей интерфейса в режиме «1С:Предприятие»

При нажатии в редакторе панелей кнопки Стандартные в интерфейсе приложения будет восстановлена настройка панелей, заданная разработчиком в конфигураторе.

Кроме того, видимость и расположение панелей интерфейса можно настроить программно при старте прикладного решения. Об этом будет рассказано в третьей части книги, в разделе «Настройка состава панелей интерфейса» на стр. 723.

Для помощи разработчику в платформе уже реализованы четыре режима основного окна приложения, которые мы рассмотрим в следующем разделе.

Режимы основного окна приложения

Установка режима основного окна – это комплексное решение, отражающее суть прикладного решения. Изменить режим основного окна можно из конфигуратора с помощью свойства Режим основного окна клиентского приложения или из встроенного языка при старте прикладного решения.

В платформе реализованы четыре режима основного окна:

- Обычный,
- Рабочее место,
- Полноэкранное рабочее место,
- Киоск.

Интерфейс режима Обычный основного окна ничем не отличается от того, что мы описывали выше.

В режиме Рабочее место скрываются все панели и область системных команд (главное меню и набор вспомогательных команд), а также надпись «Начальная страница», кнопки навигации Вперед/Назад. Кроме того, отключаются сочетания клавиш, которые используются в этих скрытых элементах интерфейса (рис. 1.14а).

Режим Рабочее место можно использовать для простых приложений, в которых реализована работа только с одной формой. Например, это может быть почтовый клиент, задача которого – принимать и отправлять почту.

В режиме Полноэкранное рабочее место скрывается то же самое, что и в режиме Рабочее место, но дополнительно к этому основное окно разворачивается на весь экран и скрываются кнопки управления окном (Свернуть, Восстановить, Закрыть), рис. 1.14б.

Режим Полноэкранное рабочее место можно использовать для создания полноэкранного рабочего места пользователя, для которого важно, чтобы на экране было минимум информации, не относящейся к его функциям. При этом ему доступна гиперссылка с именем пользователя, по которой он может завершить работу в программе.

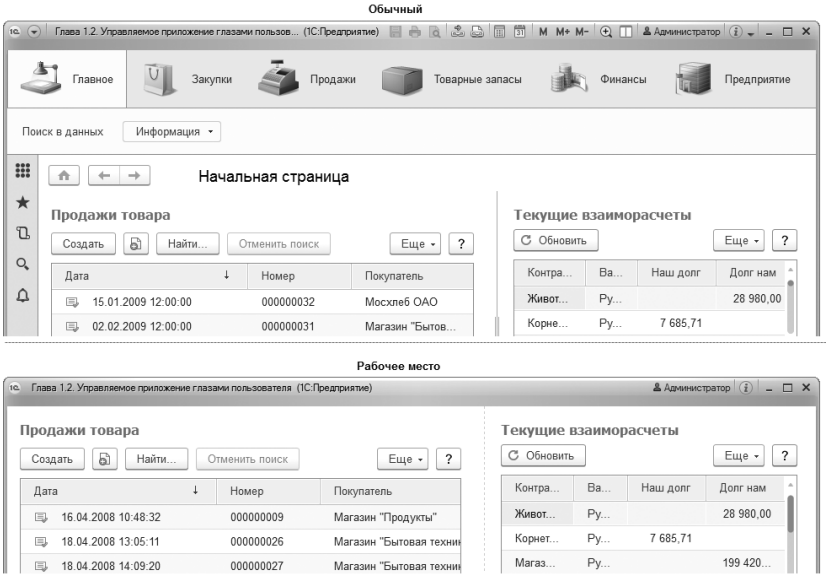


Рис. 1.14а. Сравнение режимов «Основной» и «Рабочее место» основного окна приложения

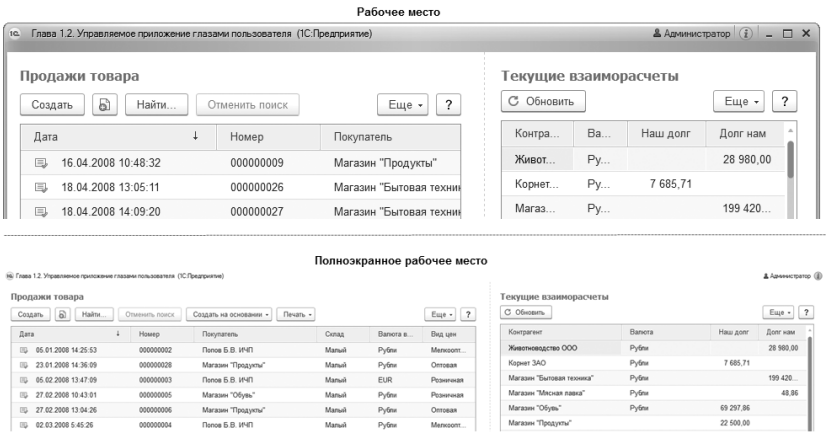


Рис. 1.14б. Сравнение режимов «Рабочее место» и «Полноэкранное рабочее место» основного окна приложения

В режиме Киоск дополнительно ко всему перечисленному скрывается гиперссылка с именем пользователя и кнопка О программе. Однако для закрытия приложения должен быть предусмотрен собственный алгоритм (рис. 1.14с).

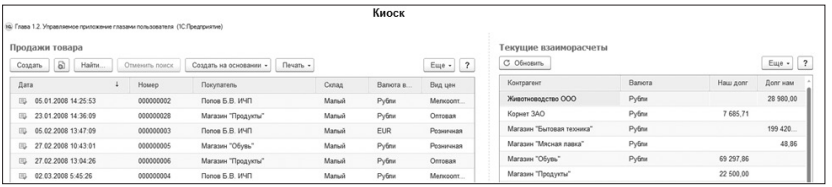


Рис. 1.14с. Режим «Киоск» основного окна приложения

Режим Киоск удобен для простых приложений, в которых не требуется аутентификация пользователя в информационной базе. Например, это может быть платежный терминал.

Для более тонкой настройки существует возможность скрывать или показывать заголовок, а также кнопку закрытия формы.

Окно клиентского приложения

Для отображения форм, созданных в прикладном решении, используются *окна клиентского приложения*. Как уже говорилось, при работе в интерфейсе «Такси» большинство форм открываются в окнах клиентского приложения, которые расположены в рабочей области основного окна приложения. Исключением являются блокирующие окна, которые могут блокировать как окно владельца, так и сразу весь интерфейс приложения.

Если окно клиентского приложения открыто в режиме блокирования окна владельца, система не позволит переключиться на окно, из которого было открыто блокирующее, до тех пор, пока последнее не будет закрыто.

Окна, блокирующие весь интерфейс, обеспечивают работу приложения без использования модальных окон. Отказ от модальности является необходимым условием для работы «1С:Предприятия» в веб-клиенте и на мобильных устройствах, а также по ряду других

причин. Подробнее об этом будет рассказано в разделе «Открытие формы в блокирующем режиме без использования модальности» на стр. 461.

При закрытии окна клиентского приложения не происходит завершения работы прикладного решения.

В окне клиентского приложения можно выделить *основную форму*, которая показывается в разделе Основное панели навигации окна клиентского приложения. При нажатии на другие ссылки в панели навигации окна клиентского приложения будут открываться *вспомогательные формы*.

Структура окна клиентского приложения следующая (рис. 1.15):

- Команды перехода между открытыми окнами клиентского приложения.
- Заголовок окна клиентского приложения.
- Панель навигации окна клиентского приложения.
- Рабочая область окна, в которой отображается форма.

Команды перехода между открытыми окнами клиентского приложения позволяют перейти к начальной странице, перемещаться вперед/назад по списку открытых окон. Рядом с ними находится кнопка в виде звездочки, с помощью которой можно поместить форму в избранное или удалить ее из избранного.

Панель навигации окна клиентского приложения позволяет просматривать различные сведения, логически связанные с данными, которые отображает основная форма окна.

В панели навигации первой всегда является команда Основное, которая предназначена для открытия основной формы окна клиентского приложения. Затем следуют остальные команды, с помощью которых можно открыть вспомогательные формы.

Если все команды не помещаются в панели навигации формы, то в самом конце панели отображается подменю Еще, которое содержит оставшиеся команды.

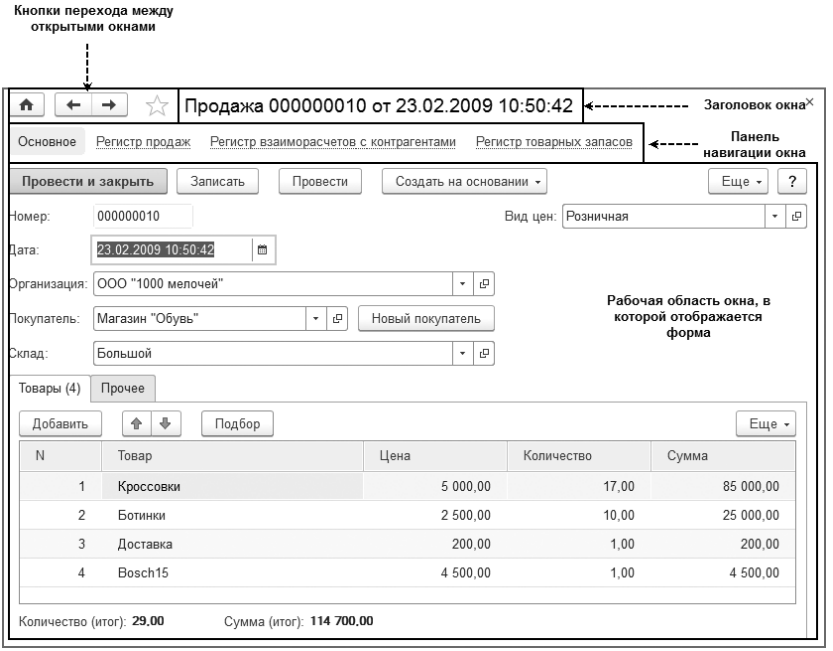


Рис. 1.15. Окно клиентского приложения прикладного решения

Командная панель формы содержит команды, непосредственно связанные с объектом, который отображается в основной форме. Команды объединены в стандартные группы Важное и Создать на основании. После них на панели могут размещаться группы команд, созданные разработчиком.

Подробнее о командном интерфейсе формы рассказывается в главе 1.12. «"Командуем" формами», стр. 184.

Если в какой-либо группе нет ни одной команды, то группа не показывается. Если все команды не помещаются в командной панели формы, то в самом конце панели отображается подменю Еще, которое содержит оставшиеся команды.

Глава 1.3. Командный интерфейс системы

В данной главе мы узнаем о командах системы «1С:Предприятие» и их источниках. Также мы разберемся, как из этих команд формируется глобальный командный интерфейс.

В качестве примера будем использовать демонстрационную базу «Глава 1.2. Прикладное решение глазами пользователя».

Команды «1С:Предприятия»

Для начала давайте поговорим о том, какие команды бывают. Все команды «1С:Предприятия» можно разделить по источникам и по области видимости (рис. 1.16).

Рис 1.16. Команды «1С:Предприятия»

ПРИМЕЧАНИЕ

Приведенная классификация не является исчерпывающей. Те же самые команды могут быть классифицированы и по другим критериям. Например, «что команды делают» – изменяют состав отображаемых данных или изменяют сами данные; «необходимость в параметре» – одни команды не требуют для своего исполнения параметров, а другим необходимо передавать параметры.

По источникам команды можно разделить на три группы:

- системные команды;
- стандартные команды объектов и расширений форм и некоторых элементов формы (таблица, табличный документ и т. д.);
- произвольные команды.

Системные команды предопределены на уровне технологической платформы и предоставляются автоматически. Их состав и выполняемые действия одинаковы для любого прикладного решения и не могут быть изменены разработчиком.

Стандартные команды предопределены на уровне объекта конфигурации (или расширения формы) и предоставляются автоматически. Состав стандартных команд зависит от того, какие объекты присутствуют в конфигурации и какие значения заданы для свойств этих объектов. Выполняемые действия также предопределены, но разработчик может в какой-то мере повлиять на них. Для этого используется механизм событий.

Произвольные команды полностью создаются разработчиком. Для этого существует объект конфигурации *Команда*. Для реализации универсального (в рамках прикладного решения) функционала используются *общие* команды. Для реализации функционала, специфического для того или иного объекта, используются команды, *подчиненные* этому объекту.

По области видимости команды делятся:

- на глобальные,
- локальные команды формы.

Глобальные команды предназначены для выбора пользователем той или иной функциональности в рамках приложения в целом. В состав глобальных команд входят *системные команды*, *стандартные команды объектов конфигурации*, а также *общие и подчиненные произвольные команды*.

В дальнейшем изложении, до главы 1.12, мы будем рассматривать глобальный командный интерфейс. При этом зачастую прилагательное «глобальный» использовать не будем.

Локальные команды формы предназначены для выполнения действий в форме. В состав локальных входят *стандартные команды* формы, *команды расширений* формы, *команды расширений некоторых элементов* формы, а также *произвольные команды* формы. Эти команды доступны только в контексте той или иной формы.

Глобальные команды формируют *глобальный командный интерфейс* прикладного решения. Локальные команды формы и, возможно, часть глобальных команд формируют *командный интерфейс форм*.

А теперь, зная, откуда берутся команды, сравним количество объектов конфигурации и форм в демонстрационной базе и количество команд, отображаемых в командном интерфейсе, например, для пользователя Продавец. Из сравнения можно сделать вывод: в командном интерфейсе мы видим не все команды, предоставляемые прикладным решением.

Каждому пользователю предоставлены только те команды, которые ему необходимы для решения своих конкретных задач.

Как же «ужимается» командный интерфейс? Чтобы ответить на этот вопрос, давайте разберемся с тем, как система формирует командный интерфейс пользователя.

На данном этапе будем считать, что для построения командного интерфейса пользователя система использует все глобальные команды. Управление составом команд, используемых для построения командного интерфейса пользователя, описано в главе 1.4, стр. 49.

Структура командного интерфейса

При запуске прикладного решения для пользователя формируется *индивидуальный глобальный командный интерфейс*. Его структура, то есть размещение команд, *однозначно определяется иерархией подсистем конфигурации и свойствами самих команд*. Состав же команд, включенных в командный интерфейс, определяется правами пользователя, значениями функциональных опций и видимостью команд по ролям.

Кроме того, если в предыдущих сеансах «1С:Предприятия» пользователь выполнял настройку командного интерфейса под себя, эти настройки также будут влиять на размещение и состав команд командного интерфейса этого пользователя.

В командном интерфейсе можно выделить несколько структурных элементов (см. главу 1.2), в каждом из которых размещаются определенные команды (рис. 1.17).

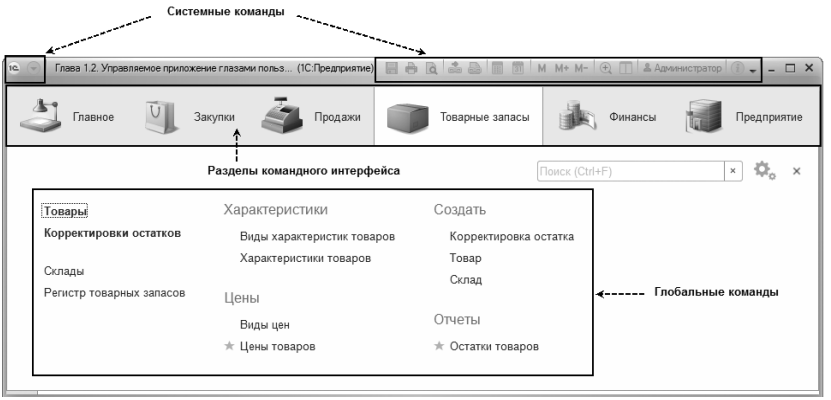


Рис. 1.17. Структура глобального командного интерфейса

Все системные команды размещаются в области системных команд основного окна приложения.

В панели разделов размещаются команды для выбора разделов, состав которых определяется структурой подсистем верхнего уровня иерархии.

При выборе раздела в панели функций или в меню функций текущего раздела показываются команды, относящиеся к выбранному разделу.

> Глобальные команды также могут быть размещены в панели навигации окна клиентского приложения и в командной панели формы.

Влияние подсистем

Почему же иерархия подсистем однозначно определяет структуру командного интерфейса? Да потому, что это логично.

Обычно с помощью подсистем описываются различные виды деятельности, автоматизируемые прикладным решением, то есть с их помощью формируется *функциональная структура* прикладного решения.

В одной подсистеме имеет смысл «собирать» объекты конфигурации, реализующие тесно связанную функциональность. При этом *состав подсистем верхнего уровня иерархии отражает наиболее общее разделение функциональности прикладного решения*. Например,

разработчик может определить подсистему Запасы и собрать в ней функции управления складскими запасами, а в подсистеме Взаиморасчеты объединить функции управления взаиморасчетами с контрагентами и т. д.

Для использования же той или иной функциональности предназначены команды. Соответственно, структурирование командного интерфейса по подсистемам представляется наиболее естественным.

Для того чтобы командный интерфейс был удобным, не перегруженным и не содержал «лишних» команд, разработку прикладного решения рекомендуется начинать с определения его функциональной структуры.

Это значит, что перед реализацией функциональности (созданием объектов конфигурации, написанием программного кода на встроенном языке) разработчик должен эту функциональность описать иерархией элементов. Корень иерархии будет соответствовать создаваемому прикладному решению (корневому элементу конфигурации), а элементы – объектам конфигурации Подсистема.

При этом платформа не накладывает никаких ограничений по количеству и уровням вложенности элементов иерархии. Это обеспечивает необходимую гибкость при описании функциональной структуры прикладного решения.

Для пользователей подсистемы верхнего уровня иерархии представляются как *разделы* глобального командного интерфейса. Команды выбора разделов размещаются в панели разделов основного окна приложения (рис. 1.18).

Выбранный раздел определяет состав команд, размещаемых в меню функций или в панели функций текущего раздела. В них отображаются команды тех объектов конфигурации, которые принадлежат соответствующей подсистеме верхнего уровня.

У подсистемы, для которой был сформирован раздел, могут существовать подчиненные подсистемы. В этом случае для каждой из подчиненных подсистем будет сформирован *подраздел*, отображаемый группой в меню функций. В подразделах отображаются команды объектов, включенных в соответствующую подчиненную подсистему.

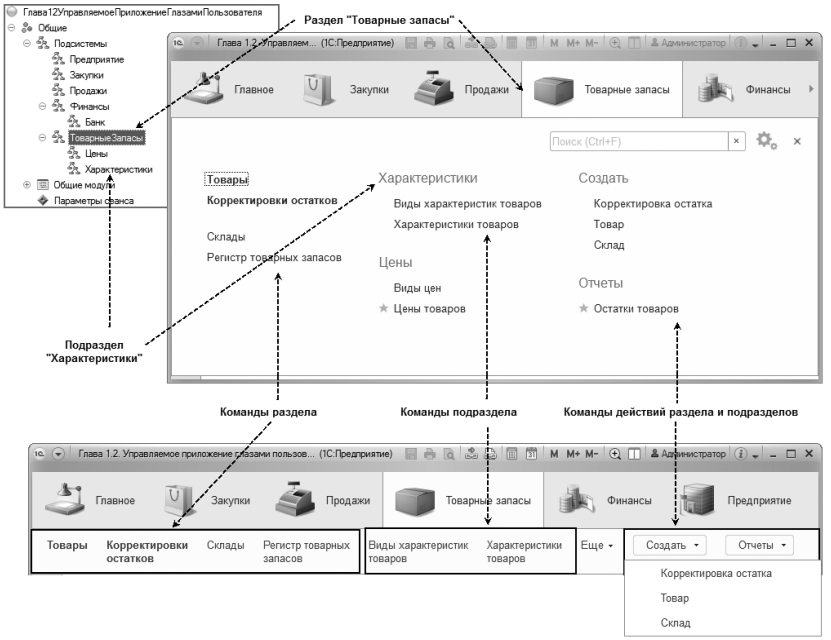


Рис. 1.18. Соответствие подсистем конфигурации и разделов командного интерфейса

Например, для раздела Товарные запасы (соответствует подсистеме ТоварныеЗапасы) в командном интерфейсе созданы подразделы Характеристики и Цены (соответствуют подчиненным подсистемам), см. рис. 1.18.

В панели функций текущего раздела для команд, принадлежащих различным подчиненным подсистемам, подразделы не создаются. Команды создания объектов, команды формирования отчетов, сервисные и др. команды в этой панели «собираются» в группы, формируемые в соответствии с назначением команд, см. рис. 1.18 (внизу).

Отдельно стоит обратить внимание на командный интерфейс раздела Главное. Этот раздел предназначен для отображения наиболее важных и наиболее часто используемых пользователем команд. Именно этот раздел активизируется при запуске прикладного решения.

В связи с тем, что для разных пользователей состав важной информации различен, раздел Главное не связан с какой-либо из подсистем.

По той же причине состав доступных команд формируется вручную разработчиком.

Для наполнения раздела Главное командами используется *редактор командного интерфейса основного раздела*. О нем мы поговорим в дальнейшем (см. раздел «Система настройки командного интерфейса» на стр. 97).

Формирование состава командного интерфейса

Разобравшись со структурированием командного интерфейса, перейдем к рассмотрению того, как платформа наполняет его командами.

Основной целью, достигаемой при формировании состава команд, является «превращение» полного командного интерфейса в командный интерфейс конкретного пользователя, запустившего прикладное решение. При этом идеология системы «1С:Предприятие» обуславливает выполнение действий по «превращению» на сервере приложений.

В соответствии с задачами, решаемыми системой при формировании командного интерфейса, процесс его построения можно условно разбить на несколько этапов:

- Обеспечение доступности только тех команд, которые пользователь имеет право выполнять.

  Для решения этой задачи используется *механизм прав доступа* – состав команд, доступных конкретному пользователю, автоматически формируется системой на основании прав, устанавливаемых для ролей этого пользователя.
- Оптимизация командного интерфейса в соответствии с задачами, решаемыми конкретным пользователем.

  Для решения этой задачи выполняется *настройка пользовательской видимости команд по ролям* – для доступных пользователю команд система определяет их видимость по умолчанию. Эта видимость задается разработчиком. При этом платформа обеспечивает возможность задания видимости в разрезе ролей.
- Исключение команд, относящихся к отключенным функциональным возможностям.

Для решения этой задачи выполняется *настройка доступности команд по функциональным опциям* – система автоматически ограничивает состав доступных команд только теми функциональными возможностями прикладного решения, которые используются в конкретном случае.

- Предоставление пользователю возможности собственной настройки видимости команд.

  Для решения этой задачи выполняется *настройка видимости команд в разрезе пользователей* прикладного решения.

Давайте кратко рассмотрим каждый из перечисленных этапов.

ВНИМАНИЕ!

> В этой главе речь идет о стандартных командах объектов конфигурации. Настройка доступности команд, созданных разработчиком, описана в главе 1.11 на стр. 149.

Обеспечение доступности команд

При формировании командного интерфейса для конкретного пользователя в первую очередь анализируются права этого пользователя на доступ к данным. Это позволяет автоматически согласовать набор команд, предоставляемых пользователю, с набором предоставленных ему прав. О системе определения прав рассказано в разделе «Система прав доступа» на стр. 65.

Доступ пользователя к разделу можно ограничить на уровне функциональной структуры прикладного решения. Для такого ограничения используется право Просмотр объекта конфигурации Подсистема.

Из командного интерфейса пользователя исключаются все разделы, соответствующие подсистемам, для которых у роли пользователя право Просмотр *не установлено.*

Например, у роли Администратор для подсистемы Закупки право Просмотр установлено. В командном интерфейсе

Как мы увидим в дальнейшем (см. раздел «Состав разделов» на стр. 51), на доступность раздела также влияет свойство подсистемы Включать в командный интерфейс. Это свойство влияет на всех пользователей.

доступна команда выбора соответствующего раздела и доступны команды этого раздела (рис. 1.19).

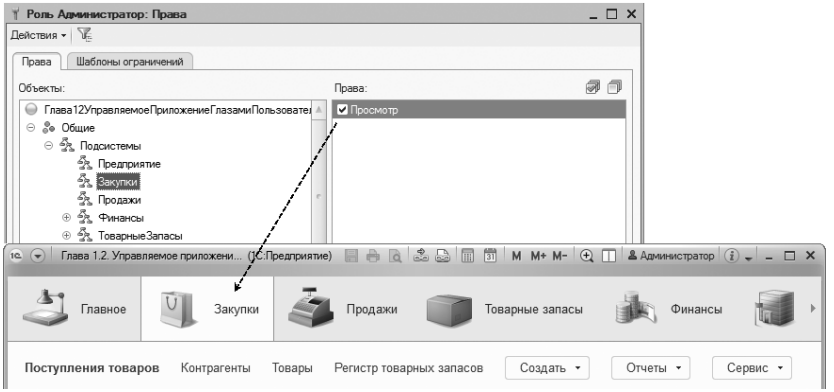


Рис. 1.19. Право на просмотр подсистемы установлено

У роли Продавец для этой же подсистемы право не установлено. В командном интерфейсе команда перехода к разделу Закупки недоступна, и невозможно использовать команды этого раздела (рис. 1.20).

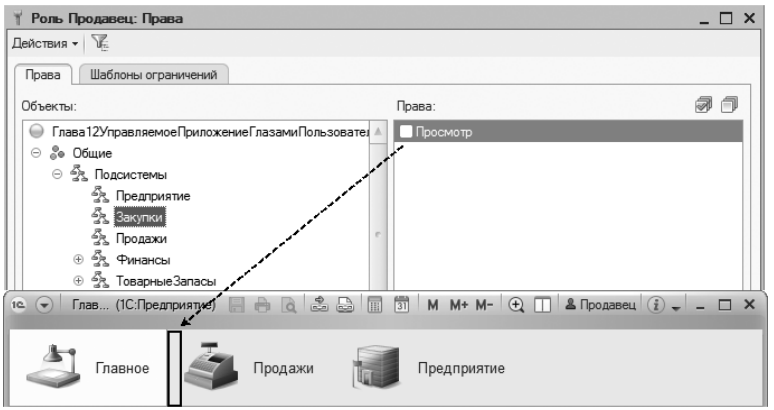


Рис. 1.20. Право на просмотр подсистемы не установлено

Если необходимо ограничить доступ пользователя только к командам, связанным с определенным объектом конфигурации, это можно реализовать установкой прав на объект конфигурации, предоставляющий эти команды.

Как мы увидим в дальнейшем (см. раздел «Стандартные команды» на стр. 56), на доступность команд объекта также влияет его свойство Использовать стандартные команды. Это свойство влияет на всех пользователей.

Из командного интерфейса пользователя исключаются стандартные команды тех объектов конфигурации, доступ к которым пользователю запрещен.

Например, у роли Администратор для справочника Контрагенты установлены все права. В командном интерфейсе доступны команды перехода к списку контрагентов и создания нового контрагента (рис. 1.21).

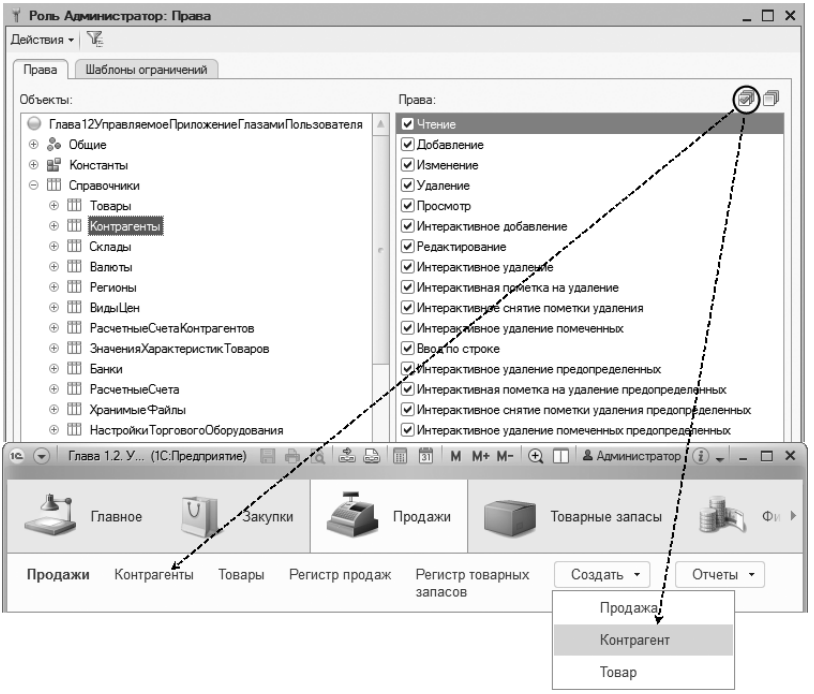


Рис. 1.21. Права на объект конфигурации установлены – команды доступны

У роли Продавец для этого же справочника ни одно право не установлено. В командном интерфейсе команд перехода к списку контрагентов и создания нового контрагента нет (рис. 1.22).

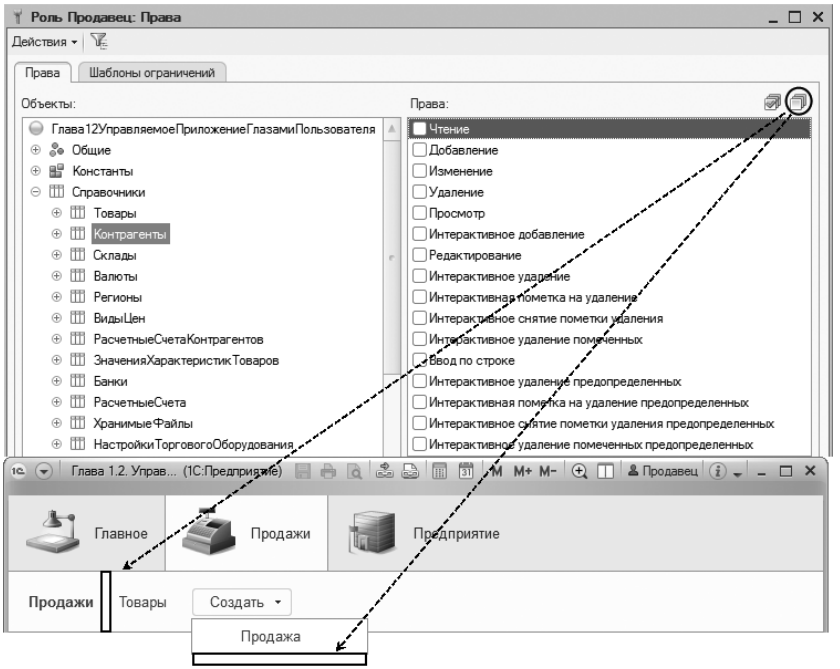


Рис. 1.22. Права на объект конфигурации сброшены – команды недоступны

Если для пользователя определен ограниченный доступ к объекту конфигурации, то из командного интерфейса исключаются только те команды объекта, которые выполняют недоступные пользователю операции. Например, у роли Администратор для справочника Товары установлены все права. В глобальном командном интерфейсе доступны команды перехода к списку товаров и создания нового товара (рис. 1.23).

У роли Продавец для этого же справочника установлены только права, позволяющие просматривать товары. В глобальном командном интерфейсе команда перехода к списку товаров доступна, а команда создания нового товара отсутствует (рис. 1.24).

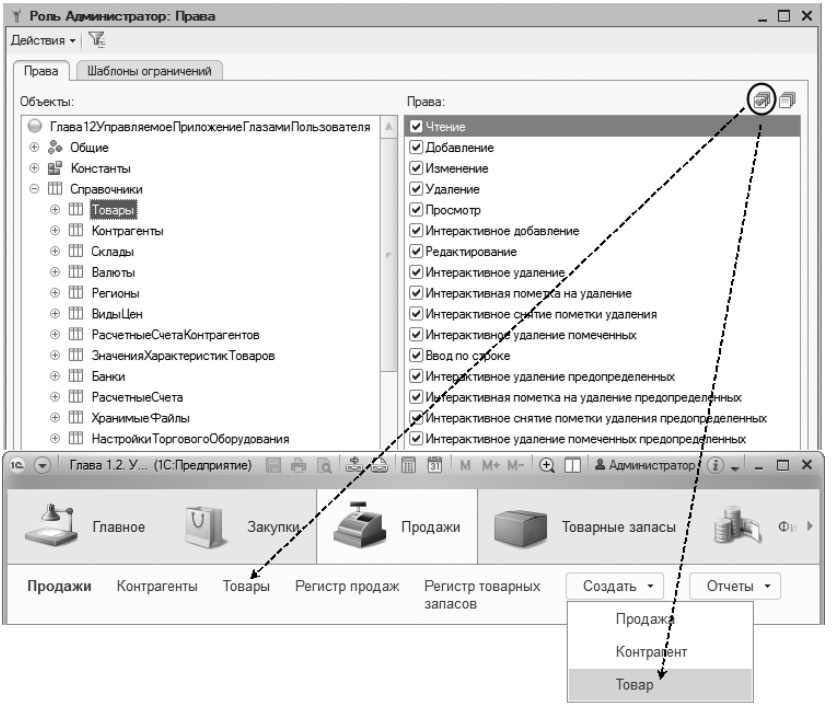


Рис. 1.23. Права на объект конфигурации установлены – команды доступны

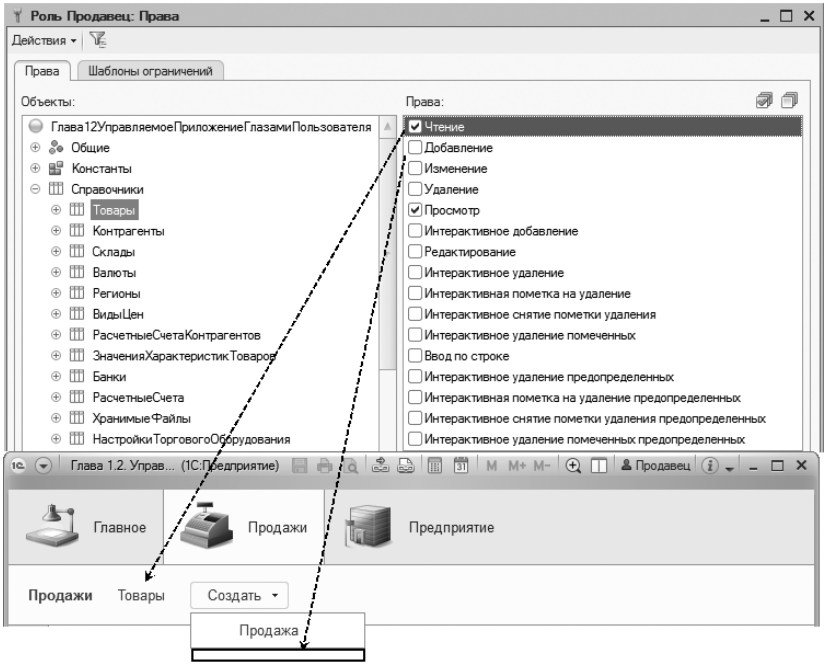


Рис. 1.24. Права на объект конфигурации установлены частично – команды доступны частично

Оптимизация командного интерфейса

Кроме доступности команд в зависимости от прав система выполняет настройку видимости доступных команд для конкретных ролей пользователей. Связано это с тем, что одна и та же доступная команда в одном разделе пользователю нужна, а в другом она только мешает.

ВНИМАНИЕ!

Необходимо различать механизм ограничения доступности на основании прав и механизм видимости по ролям. Первый жестко ограничивает пользователя в его возможностях выполнить те или иные действия. Второй позволяет оптимизировать состав видимых команд в соответствии с теми действиями, которые пользователь выполняет наиболее интенсивно.

Настройка видимости *позволяет предоставить пользователю ту часть функциональности, которая ему требуется при решении тех или иных конкретных задач*.

При этом для пользователя оставлена возможность самостоятельно донастроить видимость доступных команд – как удобно ему, а не как думает разработчик.

ВНИМАНИЕ!

> Настройка видимости выполняется только для тех команд, которые оказались доступны пользователю в соответствии с его правами (ролью). Установка видимости для недоступных команд не приведет к их появлению в командном интерфейсе.

Правила пользовательской видимости команд, автоматически устанавливаемые платформой, приведены в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 6.2.7 «Ролевая настройка видимости команд по умолчанию».

Для изменения пользовательской видимости команд по умолчанию необходимо использовать *редактор командного интерфейса*. О редакторах командного интерфейса рассказано в разделе «Система настройки командного интерфейса» стр. 97.

Например, для команды Товар: создать видимость установлена, и эта команда отображается в группе Создать панели функций текущего раздела (рис. 1.25). Для команды же Товар: создать группу видимость сброшена, и эта команда в интерфейсе не отображается.

Видимость можно настраивать как общую для всех ролей – колонка Видимость, так и для каждой роли отдельно – колонка Видимость по ролям.

Например, команда Регистр товарных запасов будет отображаться в командном интерфейсе для всех пользователей, за исключением пользователей с ролью Продавец (см. рис. 1.25).

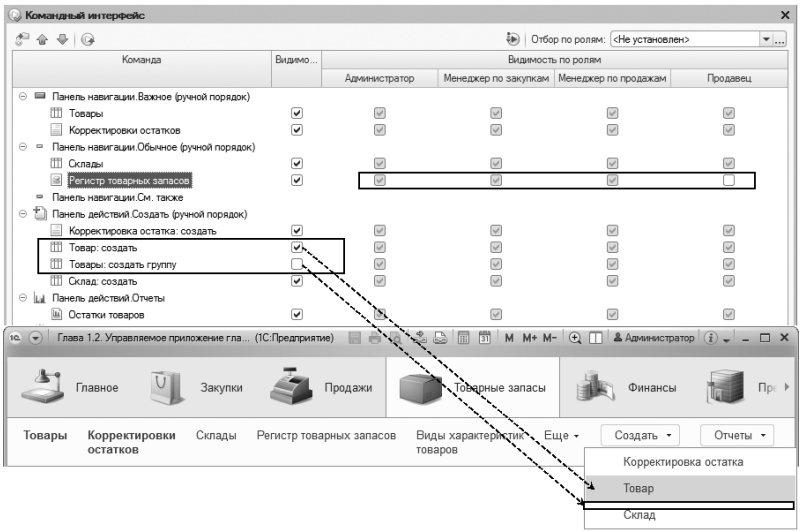


Рис. 1.25. Пользовательская видимость команд по умолчанию

Доступность команд по функциональным опциям

Установка доступности команд в зависимости от значений функциональных опций не зависит от роли пользователя, то есть выполняется одинаково для всех ролей.

Механизм функциональных опций позволяет определить в прикладном решении ту функциональность, которая может использоваться или не использоваться в зависимости от потребностей конкретной организации.

Различным элементам командного интерфейса могут быть назначены функциональные опции. При эксплуатации прикладного решения можно устанавливать значения для функциональных опций. В зависимости от установленного значения система будет автоматически включать/выключать отображение всех элементов командного интерфейса, которым эта функциональная опция назначена. О механизме функциональных опций рассказано в разделе «Механизм функциональных опций» на стр. 116.

Например, в демонстрационной базе реализован функционал учета по складам. Но часть компаний, использующих прикладное решение,

не ведут учет в разрезе складов. Для этих компаний из интерфейса необходимо исключить элементы, связанные с функционалом учета по складам.

Значение функциональной опции, управляющей учетом по складам, хранится в константе Учет по складам. Функциональная опция связана со справочником Склады.

В общем случае в качестве хранилища значений функциональных опций кроме констант могут выступать элементы справочников и записи регистра сведений.

Если константа имеет значение Истина, то команды для работы со справочником включены в командный интерфейс (рис. 1.26).

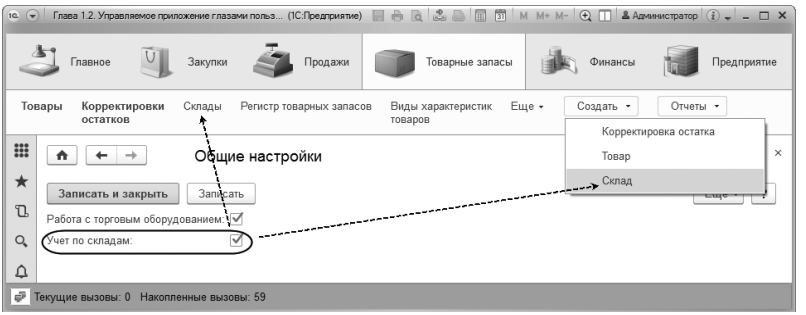


Рис. 1.26. Команды доступны при включенной функциональной опции

Если же значение константы Ложь, то эти команды из командного интерфейса исключены (рис. 1.27).

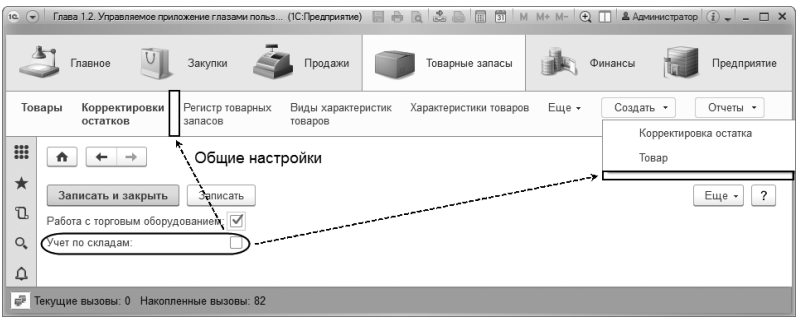


Рис. 1.27. Команды недоступны при отключенной функциональной опции

Таким образом, при отсутствии учета в разрезе складов мы имеем возможность убрать из прикладного решения «лишнюю» функциональность, в том числе и команды из командного интерфейса.

Пользовательская настройка командного интерфейса

При формировании видимой части командного интерфейса система осуществляет установку видимости команд в зависимости от персональных настроек пользователя. Этот *механизм позволяет пользователям настраивать командный интерфейс в соответствии со своими предпочтениями*. О механизме настроек пользователя рассказано в главе 1.8 на стр. 126.

Например, при первом запуске демонстрационной базы от имени пользователя Администратор состав видимых команд соответствует настройкам, выполненным разработчиком. Все подсистемы верхнего уровня отображаются в виде разделов прикладного решения. Выбрав раздел Товарные запасы и открыв меню функций, мы видим полный состав команд текущего раздела (рис. 1.28).

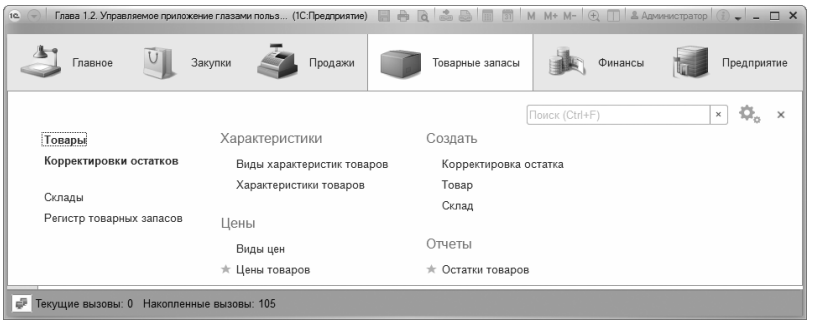


Рис. 1.28. Командный интерфейс при первом запуске

Затем пользователь изменил состав отображаемых разделов (рис. 1.29) – из списка выбранных удалил разделы Закупки и Предприятие.

Затем пользователь изменил состав команд раздела Товарные запасы. Из списка выбранных удалил команды Корректировки остатков, Регистр товарных запасов, Характеристики товаров, Цены товаров (рис. 1.30).

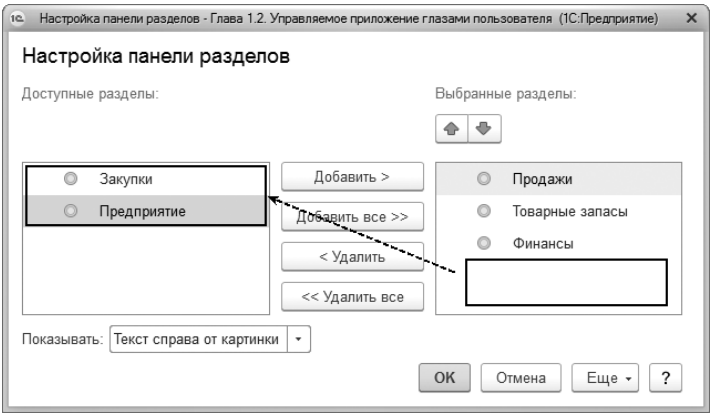


Рис. 1.29. Пользовательская настройка разделов прикладного решения

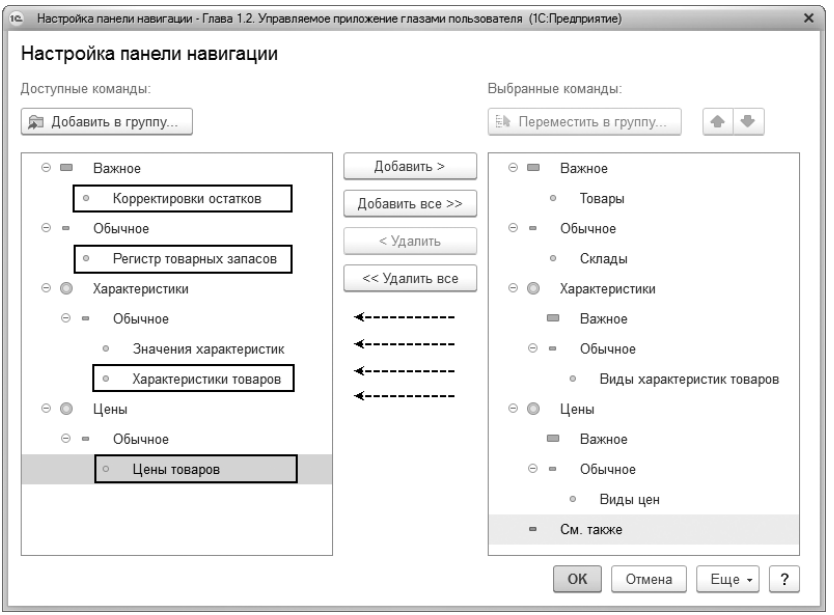


Рис. 1.30. Настройка команд раздела «Товарные запасы»

Также из группы Создать удалил команду Корректировка остатка (рис. 1.31).

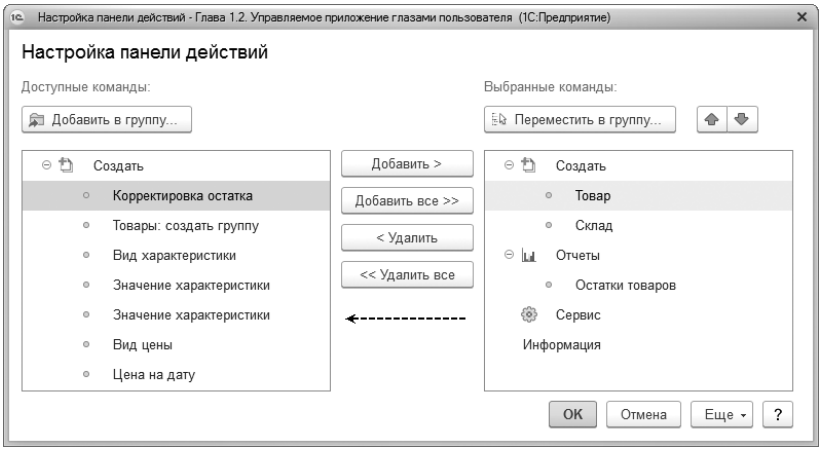


Рис. 1.31. Настройка команд раздела «Товарные запасы»

После применения всех настроек (и при последующих запусках) состав видимых команд соответствует настройкам пользователя Администратор (рис. 1.32) – все скрытые пользователем команды не отображаются.

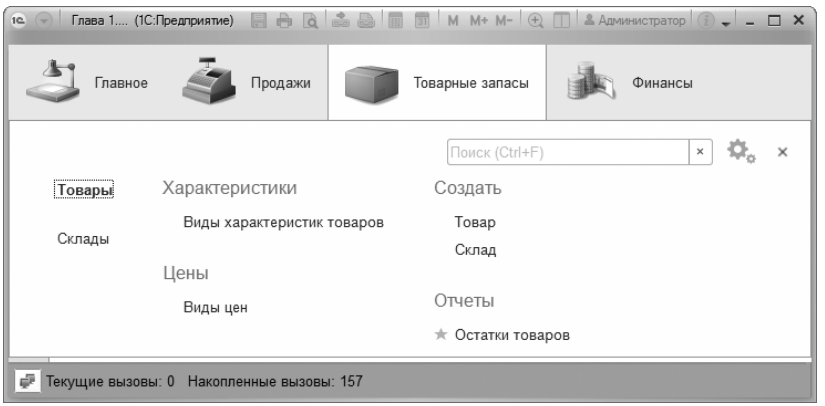


Рис. 1.32. «Суженный» пользователем командный интерфейс

Важно помнить, что отключенные команды остались доступны: пользователь вновь может включить их отображение.

Краткие итоги

В заключение рассмотрения процесса построения глобального командного интерфейса можно представить изложенный материал в виде схемы (рис. 1.33).

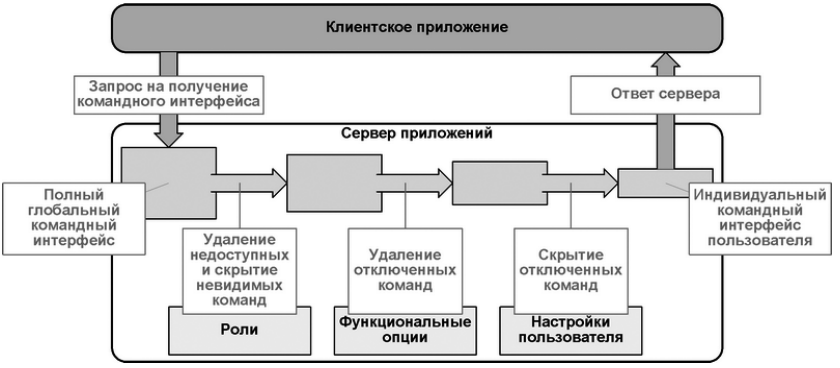


Рис. 1.33. Построение глобального командного интерфейса

В общем случае формирование глобального командного интерфейса выполняется в зависимости:

- от структуры подсистем и их свойств Включать в командный интерфейс – определяют структуру командного интерфейса;
- состава объектов конфигурации и их свойств Использовать стандартные команды – определяют состав команд прикладного решения;
- прав пользователя, определяющих состав доступных команд;
- настройки пользовательской видимости команд по умолчанию (возможно, в разрезе ролей);
- установленных значений функциональных опций;
- настроек самого пользователя.

На рис. 1.33 величина прямоугольника, представляющего создаваемый глобальный командный интерфейс, отражает его «насыщенность» командами.

Глава 1.4. Настраиваем состав команд

В дальнейшем изложении для демонстрации механизмов конструирования командного интерфейса мы будем использовать демонстрационную базу «Глава 1.2. Прикладное решение глазами пользователя».

В качестве демонстрационного примера мы будем решать практическую задачу по адаптации этой конфигурации. Результат решения этой задачи содержится в демонстрационной базе «Глава 1.4. Настраиваем состав команд».

Постановка задачи

На предприятии процесс ценообразования выделен в отдельный функциональный блок и управлением ценами занимаются специально выделенные сотрудники. В демонстрационной базе операции управления ценами совмещены с операциями управления товарными запасами в подсистеме Товарные запасы.

Необходимо выделить функционал управления ценами в отдельную подсистему и создать командный интерфейс для сотрудников, работающих с этой подсистемой.

> В данной главе мы будем рассматривать системные и стандартные команды. Произвольные команды рассматриваются в главе 1.11 на стр. 149.

Состав разделов

Как уже было сказано, состав команд панели разделов определяется составом объектов конфигурации Подсистема верхнего уровня иерархии.

Однако создание в дереве конфигурации подсистемы верхнего уровня еще не гарантирует, что для нее при построении командного интерфейса будет создан раздел.

ПРИМЕЧАНИЕ

Для простых конфигураций возможна работа без подсистем. В этом случае все команды располагаются в панели функций или в меню функций раздела Главное. Этот раздел формируется всегда. Однако в зависимости от настроек он может быть пустым. При отсутствии подсистем в конфигурации панель разделов и сама закладка раздела Главное не показываются.

Формирование раздела командного интерфейса для подсистемы верхнего уровня иерархии управляется свойством подсистемы Включать в командный интерфейс (рис. 1.34).

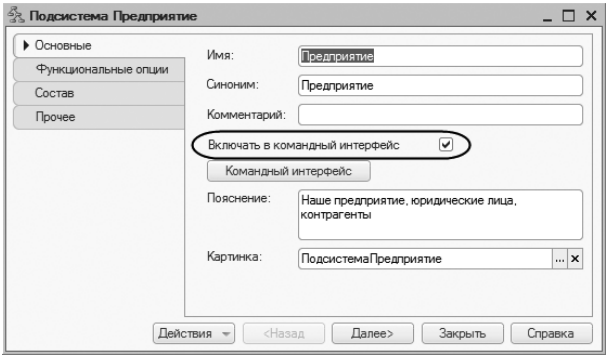


Рис. 1.34. Управление формированием раздела для подсистемы

Данное свойство оказывает влияние на формирование раздела командного интерфейса для всех пользователей прикладного решения, независимо от назначенных им ролей. Доступность же раздела для того или иного пользователя определяется назначенной ему ролью (см. раздел «Обеспечение доступности команд» на стр. 38).

Если для подсистемы верхнего уровня свойство Включать в командный интерфейс установлено, то для нее будет

В «1С:Предприятии» подсистемы могут использоваться также «исключительно в целях разработки», т. е. для группировки/фильтрации объектов конфигурации в режиме Конфигуратор. При таком использовании у подсистемы свойство Включать в командный интерфейс обычно сбрасывается.

сформирован отдельный раздел. Стандартные команды объектов конфигурации, включенных в подсистему, будут учитываться при формировании командного интерфейса пользователя.

Например, для подсистемы Финансы свойство Включать в командный интерфейс установлено (рис. 1.35). Для этой подсистемы сформирован раздел и команды включены в командный интерфейс.

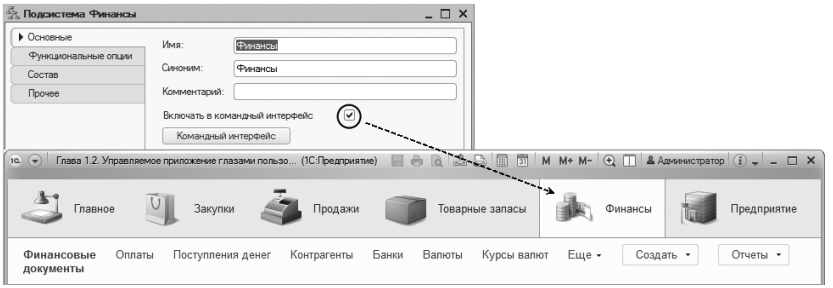


Рис. 1.35. В командном интерфейсе раздел формируется для подсистем с установленным свойством «Включать в командный интерфейс»

Если же для подсистемы свойство Включать в командный интерфейс сброшено, то стандартные команды объектов, принадлежащих этой подсистеме, не будут учитываться при формировании командного интерфейса пользователя.

Например, для подсистемы Финансы свойство Включать в командный интерфейс сброшено (рис. 1.36). Для этой подсистемы раздел не сформирован и, следовательно, команды недоступны.

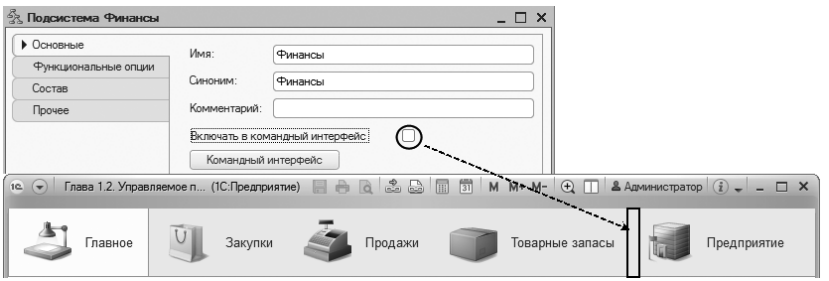


Рис. 1.36. Свойство «Включать в командный интерфейс» сброшено

Теперь мы можем начать решать поставленную задачу.

Прежде всего создадим новую подсистему верхнего уровня иерархии. Для добавления подсистемы используем команду Добавить контекстного меню узла Подсистемы дерева конфигурации. В результате будет добавлен объект конфигурации и откроется окно редактирования подсистемы (рис. 1.37).

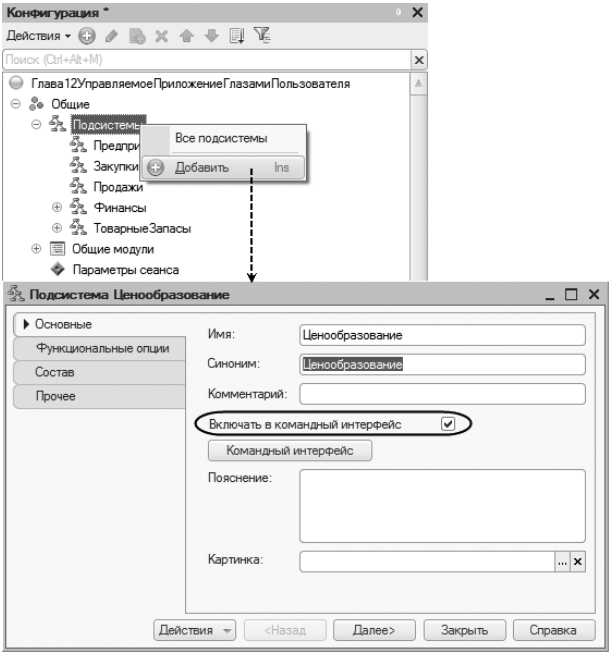


Рис. 1.37. Новая подсистема «Ценообразование» и ее свойства

На закладке Основные заполним поля:

- Имя – определяет идентификатор объекта для доступа из встроенного языка; зададим имя Ценообразование.
- Синоним – определяет представление объекта в интерфейсе; оставим автоматически сформированный синоним.

Про настройку представлений объектов конфигурации более подробно рассказывается в главе 1.9 на стр. 140.

Теперь необходимо вспомнить, что доступность раздела для пользователя определяется значением права Просмотр, установленным у роли пользователя для соответствующей подсистемы. Проконтролируем значение этого права для роли Администратор (рис. 1.38) – право должно быть установлено.

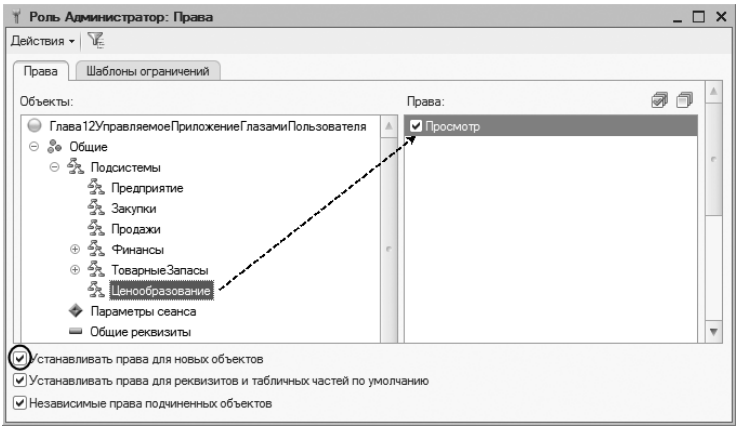


Рис. 1.38. Для роли «Администратор» установлено право «Просмотр» на подсистему «Ценообразование»

Это происходит автоматически, так как у роли Администратор установлен флажок Устанавливать права для новых объектов.

Следующей операцией по созданию раздела является его наполнение командами.

Контроль значения права осуществляется в редакторе прав. О работе с этим редактором рассказано в разделе «Система прав доступа» на стр. 65.

ВНИМАНИЕ!

Если в подсистему не включено ни одного объекта конфигурации, раздел в командном интерфейсе не показывается.

Стандартные команды

Для начала разберемся, какими же командами *может быть* наполнен раздел командного интерфейса.

В раздел командного интерфейса включаются стандартные команды тех объектов конфигурации, которые принадлежат соответствующей подсистеме верхнего уровня (и подсистемам, иерархически ей подчиненным).

Что же мы подразумеваем под стандартной командой? Каждый объект конфигурации имеет стандартный набор свойств, определяющий его поведение по умолчанию. Для доступа к функциональности объекта платформа предоставляет некоторый набор команд. Вот эти команды и являются стандартными.

ВНИМАНИЕ!

Для стандартных команд выполняемые действия определены на уровне технологической платформы и не могут быть изменены разработчиком.

Например, в демонстрационной базе справочник Товары включен в подсистемы Предприятие, Закупки, Продажи и Товарные запасы (рис. 1.39). Стандартные команды этого объекта доступны в разделах командного интерфейса, сформированных для соответствующих подсистем.

А от чего зависит состав стандартных команд? Конечно же, от класса объектов, которому принадлежит объект конфигурации. И это логично, так как различные классы объектов конфигурации предназначены для решения различных прикладных задач.

Например, справочник Товары предоставляет команды открытия списка элементов – Товары, создания нового элемента – Товар и создания новой

Непосредственно сами стандартные команды не имеют событий, для которых разработчик мог бы описать собственные алгоритмы. Но в результате выполнения многих стандартных команд вызываются события, связанные с данными, формами или элементами форм. Поэтому разработчик, используя механизм событий, может в некоторой степени влиять на действия, выполняемые стандартными командами.

группы – Товар: создать группу, а отчет Остатки товаров только команду открытия формы отчета – Остатки товаров (рис. 1.40).

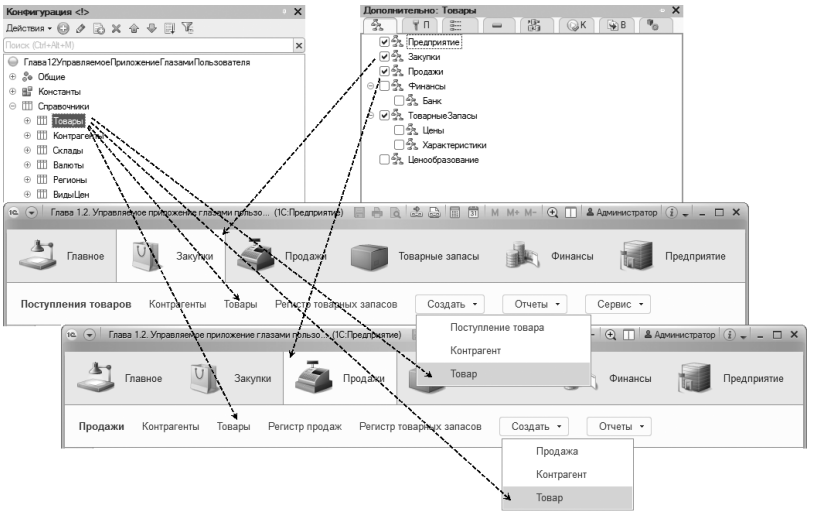


Рис. 1.39. Справочник «Товары» принадлежит нескольким подсистемам – стандартные команды объекта доступны в соответствующих разделах

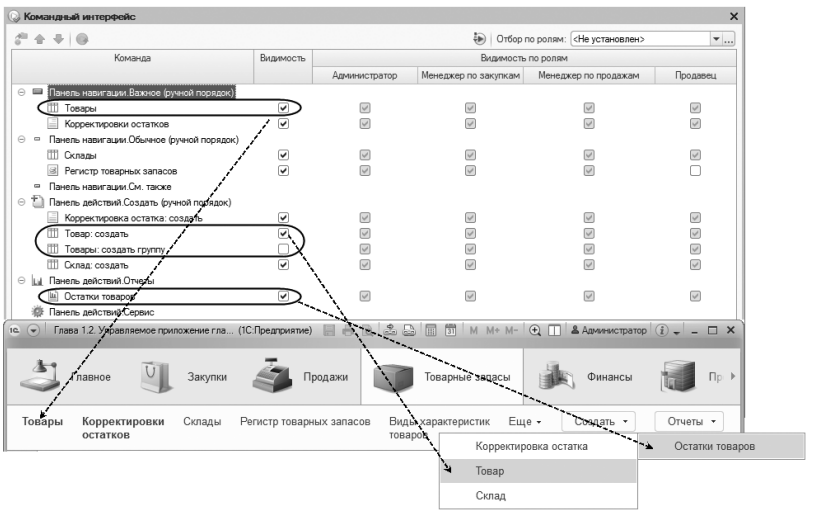


Рис. 1.40. Состав стандартных команд зависит от прототипа объекта конфигурации

Например, справочник Товары предоставляет команды открытия списка элементов – Товары, создания нового элемента – Товар и создания новой группы – Товар: создать группу, а отчет Остатки товаров только команду открытия формы отчета – Остатки товаров (рис. 1.40).

Состав предоставляемых стандартных команд мы видим в окне редактора командного интерфейса. О работе с этим редактором рассказано в разделе «Система настройки командного интерфейса» на стр. 97.

Стандартная команда Товар: создать группу по умолчанию невидима в командном интерфейсе, и поскольку мы ее видимость не включали, то она не отображается среди команд выбранного раздела.

Полный список командообразующих объектов конфигурации и предоставляемые ими стандартные команды приведены в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 6.1.2.1.

Кроме прототипа на состав команд объекта конфигурации влияют и значения свойств этого объекта.

Например, справочник Товары предоставляет стандартные команды как для создания элемента, так и для создания группы, а справочник Склады – только команду создания нового элемента (рис. 1.41).

А причина заключается в том, что у справочника Товары установлено свойство Иерархический с иерархией групп и элементов, а у справочника Склады свойство Иерархический сброшено. Таким образом, справочник Склады не может содержать записей-групп, и, следовательно, команда создания группы для этого объекта конфигурации не имеет смысла.

Так же как и с подсистемами, наличие объекта в дереве конфигурации еще не гарантирует наличия его команд в командном интерфейсе. Возможность использовать стандартные команды объекта конфигурации при построении командного интерфейса определяется значением свойства Использовать стандартные команды этого объекта.

При установленном свойстве команды объекта доступны для включения в командный интерфейс.

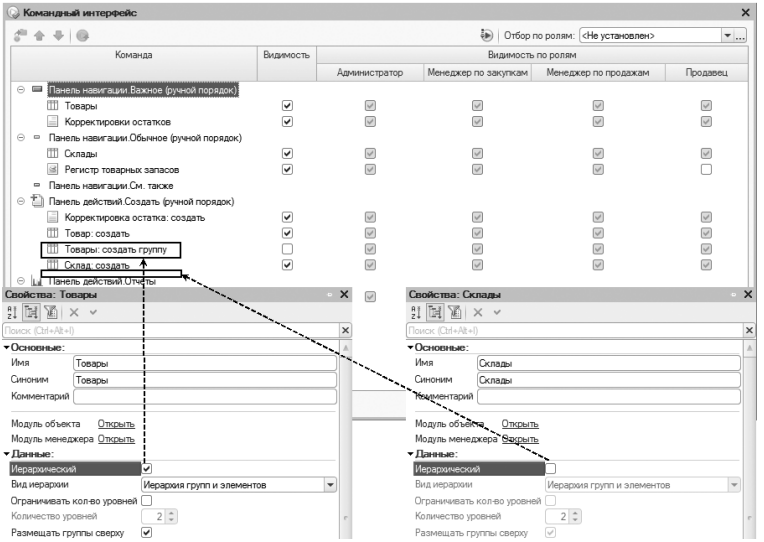


Рис. 1.41. Состав стандартных команд зависит от настройки свойств объекта конфигурации

Например, свойство Использовать стандартные команды для справочника Товары установлено (рис. 1.42). В командном интерфейсе присутствует команда Товары, которая позволяет открыть форму списка справочника, и команда Товар, которая позволяет создать новый элемент справочника.

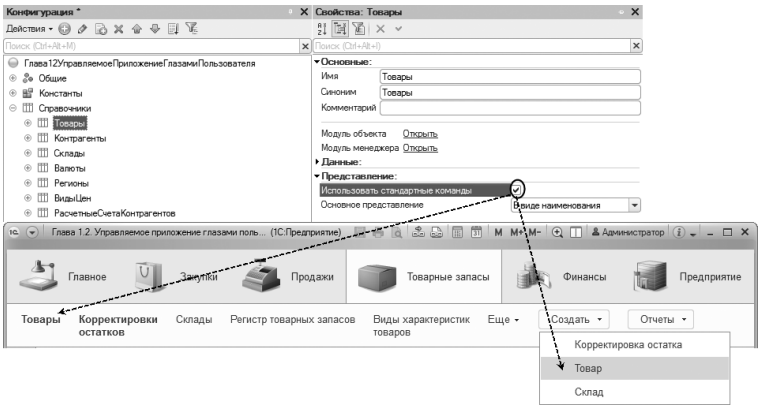


Рис. 1.42. Свойство «Использовать стандартные команды» установлено

Если же для объекта конфигурации свойство Использовать стандартные команды сброшено, то стандартные команды этого объекта в любом случае не будут использоваться при построении командного интерфейса.

Например, при сброшенном свойстве для справочника Товары в командном интерфейсе отсутствуют команды открытия списка справочника и создания нового элемента (рис. 1.43).

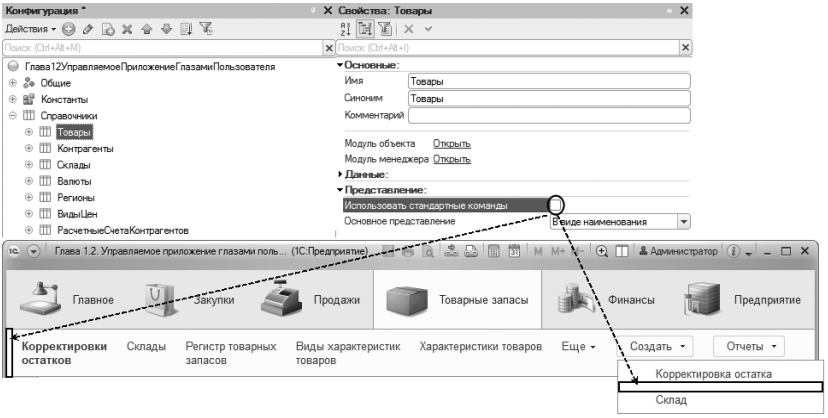


Рис. 1.43. Свойство «Использовать стандартные команды» сброшено

Для вновь создаваемого объекта конфигурации свойство Использовать стандартные команды по умолчанию установлено.

Значение свойства Использовать стандартные команды объекта конфигурации не влияет на использование произвольных подчиненных команд объекта (см. раздел «Произвольные команды» на стр. 149).

Пришло время обеспечить командами подсистему Ценообразование. Для начала определимся, какими прикладными объектами конфигурации реализован функционал управления ценами. В нашей конфигурации их три:

- Справочник Товары – хранит список товаров и услуг, для которых могут назначаться цены.

- Справочник Виды цен – хранит список видов цен, которые могут быть назначены каждому товару (услуге).
- Регистр сведений Цены товаров – хранит значения цен, назначенных товарам в разрезе различных видов цен.

Включим каждый из указанных объектов в нашу подсистему. Для этого необходимо заполнить свойство Состав подсистемы Ценообразование.

Откроем окно редактирования свойств нашей подсистемы, перейдем на закладку Состав и отметим требуемые объекты конфигурации (рис. 1.44).

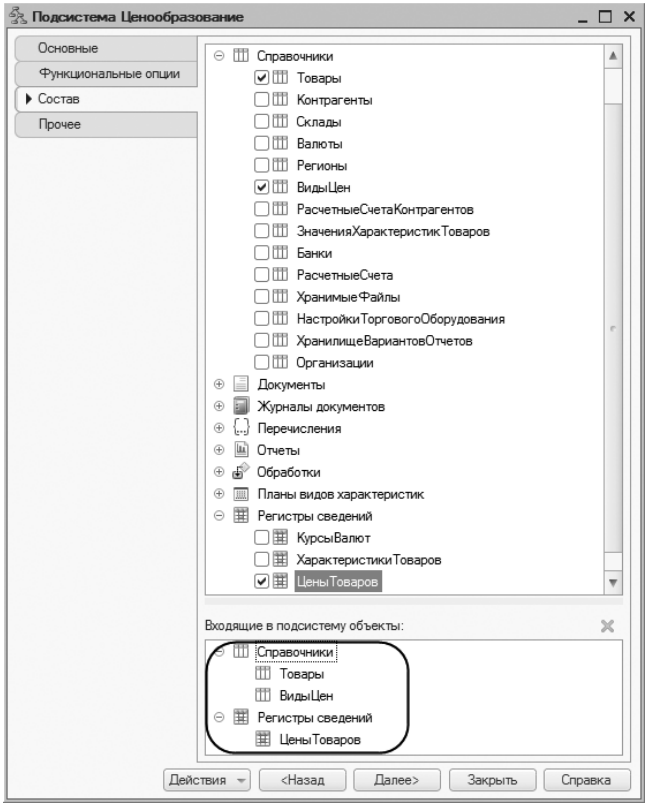


Рис. 1.44. Включение объектов конфигурации в подсистему

Для каждого из включенных объектов проверим свойство Использовать стандартные команды – оно должно быть установлено.

Как и для подсистемы, доступность для пользователя объектов и их стандартных команд определяется значением прав, установленных у роли пользователя для этих объектов. Проконтролируем значения права (на примере справочника Товары) у роли Администратор (рис. 1.45) – права должны быть установлены.

Контроль прав на справочник Виды цен и регистр сведений Цены товаров выполняем аналогично справочнику Товары.

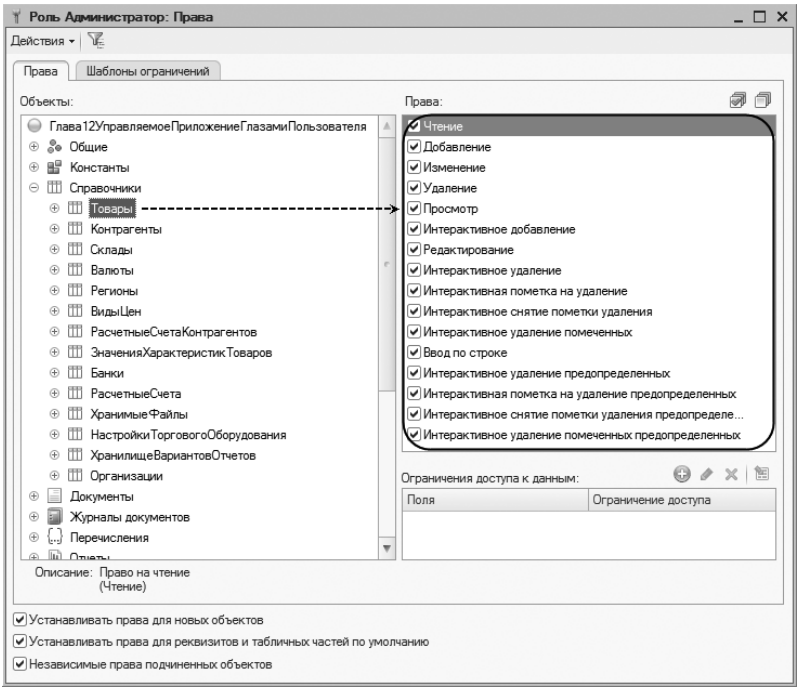


Рис. 1.45. Для роли «Администратор» установлены все права на справочник «Товары»

Сохраним внесенные изменения, запустим демонстрационную базу в режиме 1С:Предприятие от имени пользователя Администратор и в панели разделов основного окна выберем раздел Ценообразование (рис. 1.46).

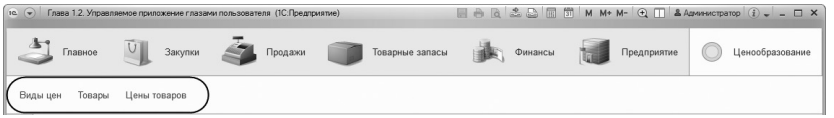


Рис. 1.46. Раздел «Ценообразование» с командами

В меню функций раздела представлены команды открытия списков объектов, принадлежащих подсистеме Ценообразование. Представление стандартных команд объектов сформировано системой на основании значений в свойстве Представление списка.

На данном этапе мы сформировали структуру подсистемы Ценообразование и обеспечили доступ к ней пользователям с ролью Администратор. По условию задачи управлением ценами будут заниматься специально выделенные сотрудники, а никак не администраторы системы. Поэтому нам необходимо для этих сотрудников создать отдельную роль. Этим мы займемся в следующей главе.

Настройкой представления мы займемся в главе 1.9 на стр. 140.

Основные действия для создания командного интерфейса

Из приведенных примеров видно, что для построения командного интерфейса необходимо и достаточно выполнить следующие действия:

- Создать иерархию подсистем.
- При необходимости установить свойство подсистемы Включать в командный интерфейс.
- Установить значение права Просмотр на подсистемы для тех ролей, которым должны отображаться соответствующие разделы командного интерфейса.
- Определить состав объектов каждой подсистемы.
- При необходимости установить свойство Использовать стандартные команды для объектов конфигурации, включенных в подсистемы.

На основании этой информации «1С:Предприятие» автоматически построит командный интерфейс пользователя. Пользователю будет отображена функциональная структура прикладного решения (разделы и подразделы) и предоставлен доступ к стандартной функциональности прикладных объектов (стандартные команды).

Глава 1.5. Настраиваем доступность команд по ролям

В данной главе мы рассмотрим систему прав доступа и ее влияние на командный интерфейс. В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.4. Настраиваем состав команд». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.5. Настраиваем доступность команд по ролям».

Система прав доступа

Основным механизмом настройки доступности команд в командном интерфейсе является *система прав доступа*.

Операция назначения роли пользователю решает две основные задачи:

- Во-первых, ограничивает состав пользователей, имеющих доступ к конфиденциальной информации.
- Во-вторых, предотвращает возможные потери информации путем запрета выполнения пользователем определенных операций (в первую очередь операций удаления и корректировки данных).

При проектировании прикладного решения разработчик должен выделить группы пользователей, выполняющих одинаковые и/или логически связанные операции. Для каждой из выделенных групп разработчик добавляет собственную роль (рис. 1.47). Каждая из ролей определяет независимый (от других ролей) набор прав доступа к обрабатываемой в прикладном решении информации и действий, которые пользователь может выполнять с этой информацией.

Разработчик имеет возможность создать любое необходимое количество ролей, обеспечивающих предоставление разным пользователям различных полномочий.

Используя механизмы управления пользователями, администратор системы регистрирует пользователей прикладного решения. Каждому пользователю назначается роль (или несколько ролей), которая определяет права пользователя на доступ к данным.

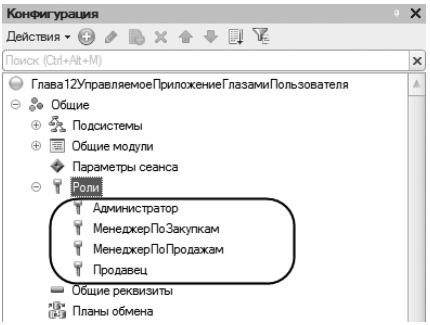


Рис. 1.47. Объекты конфигурации с шаблоном «Роли»

Настройка прав выполняется в окне редактирования роли, которое открывается двойным щелчком мыши на выбранной роли (рис. 1.48).

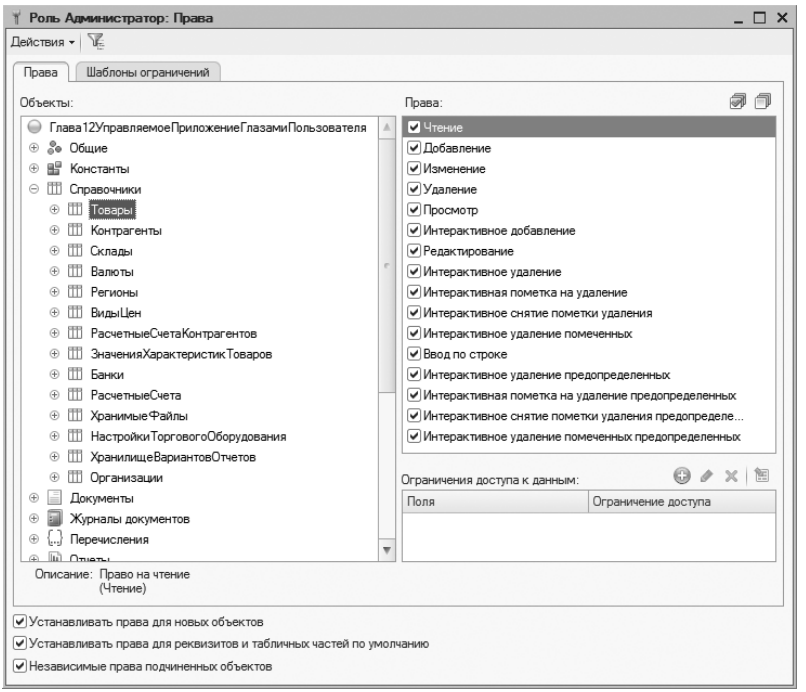


Рис. 1.48. Окно редактирования прав для роли

Для настройки прав необходимо в дереве конфигурации выбрать объект (левое поле) и в списке прав (правое поле) установить или снять отметку рядом с настраиваемым правом. С помощью кнопок в правом верхнем углу над списком прав можно установить или снять сразу все права (см. рис. 1.49).

Права можно установить на уровне классов объектов конфигурации. Для этого в дереве конфигурации необходимо выбрать узел, представляющий соответствующий класс (рис. 1.49).

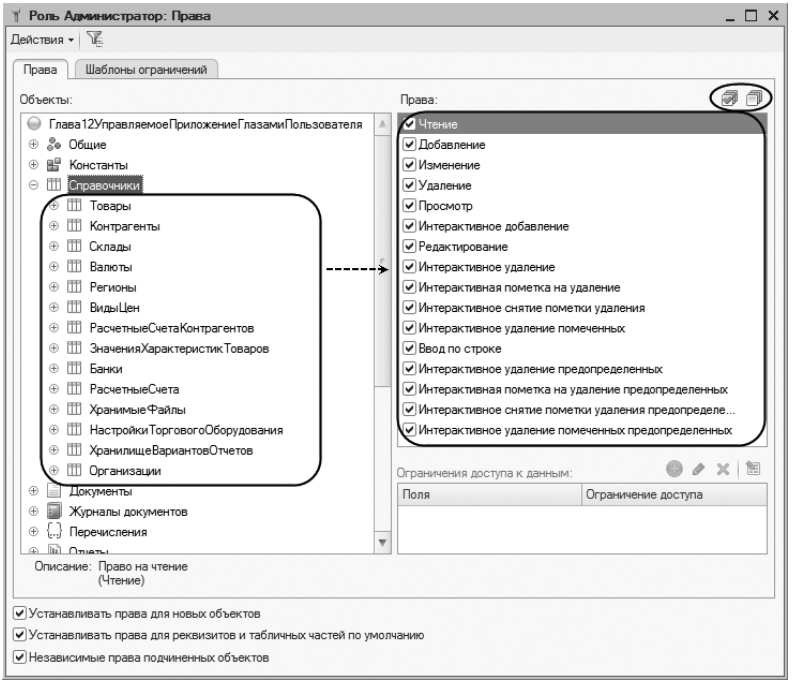


Рис. 1.49. Установка прав для группы однотипных объектов конфигурации

В этом случае назначаются одинаковые права всем объектам конфигурации этого класса. Например, на рис. 1.49 полные права будут установлены сразу для всех справочников.

Права могут быть установлены и на отдельный объект конфигурации (рис. 1.50).

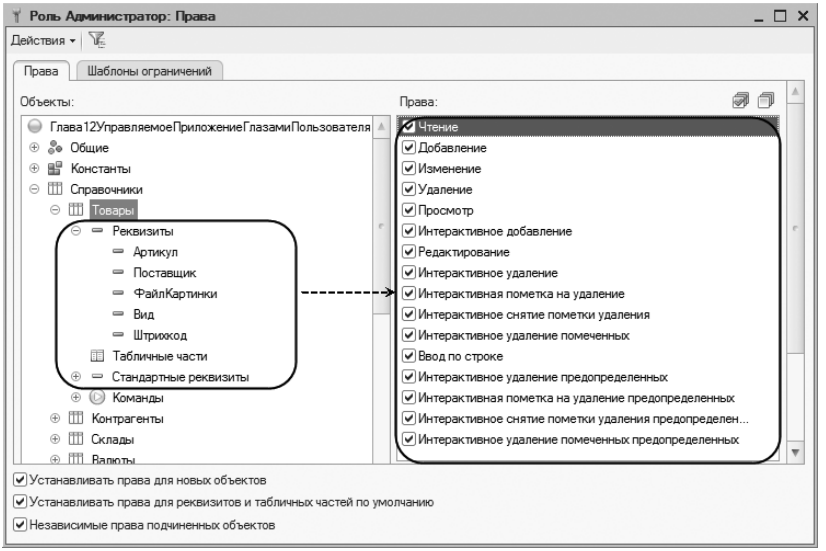


Рис. 1.50. Установка прав на объект конфигурации

В этом случае права применяются к объектам данных информационной базы, которые описаны выбранным объектом конфигурации.

Права также могут быть установлены на объекты, подчиненные выбранному (как в целом для всех подчиненных объектов одного типа, так и индивидуально на каждый из подчиненных объектов) – реквизиты, табличные части и команды (рис. 1.51).

В этом случае права применяются только к командам, подчиненным объекту, или к данным, хранящимся в подчиненных информационных структурах.

Кроме прав, применяемых для объектов конфигурации, существует набор прав, применяемый к конфигурации в целом. Список этих прав отображается при выборе корневого узла дерева конфигурации (рис. 1.52).

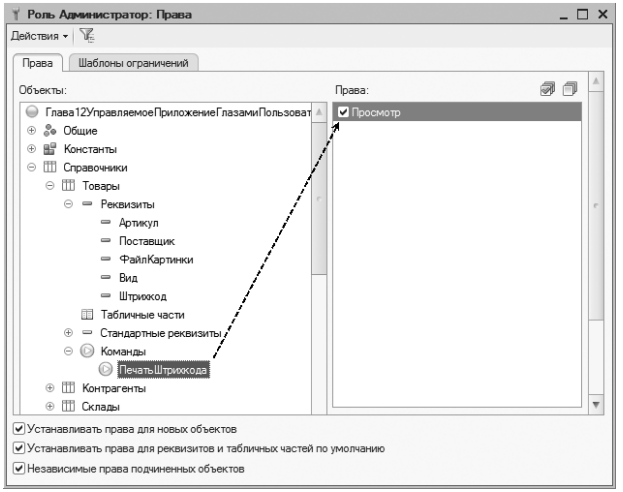


Рис. 1.51. Установка прав на подчиненный объект конфигурации

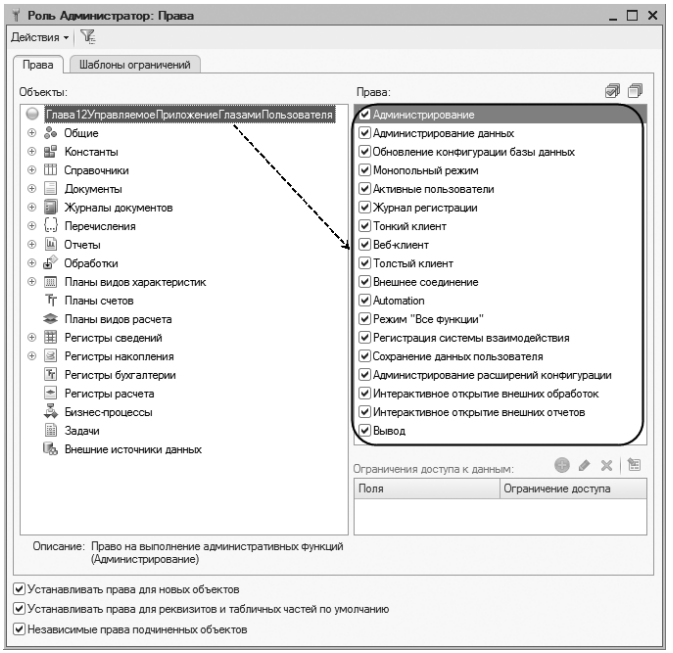


Рис. 1.52. Набор прав конфигурации

Эти права определяют возможность выполнения административных функций, использования внешних отчетов и обработок, доступа к журналу регистрации, сохранения данных и настроек пользователя и др.

А теперь давайте создадим роль Менеджер по ценам. Эта роль будет назначена пользователям, которые выполняют операции по управлению ценами.

Для добавления роли используем команду Добавить контекстного меню узла Роли дерева конфигурации. В результате будет добавлен объект конфигурации и откроется палитра свойств новой роли.

В группе Основные зададим значения свойств роли (рис. 1.53):

- Имя – МенеджерПоЦенам;
- Синоним – оставим автоматически сформированный синоним;
- Комментарий – не будем заполнять.

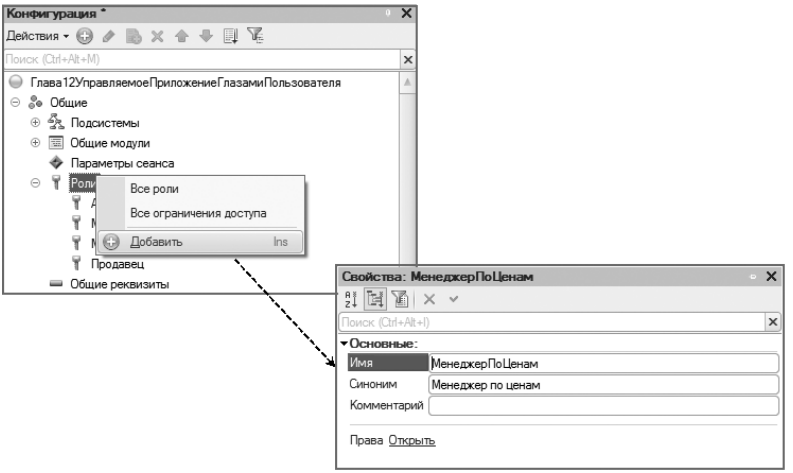


Рис. 1.53. Добавление новой роли

Одновременно откроется окно редактирования прав роли МенеджерПоЦенам. Стандартно для новой роли уже установлены права на конфигурацию в целом: Тонкий клиент, Веб-клиент, Сохранение данных пользователя и Вывод (рис. 1.54).

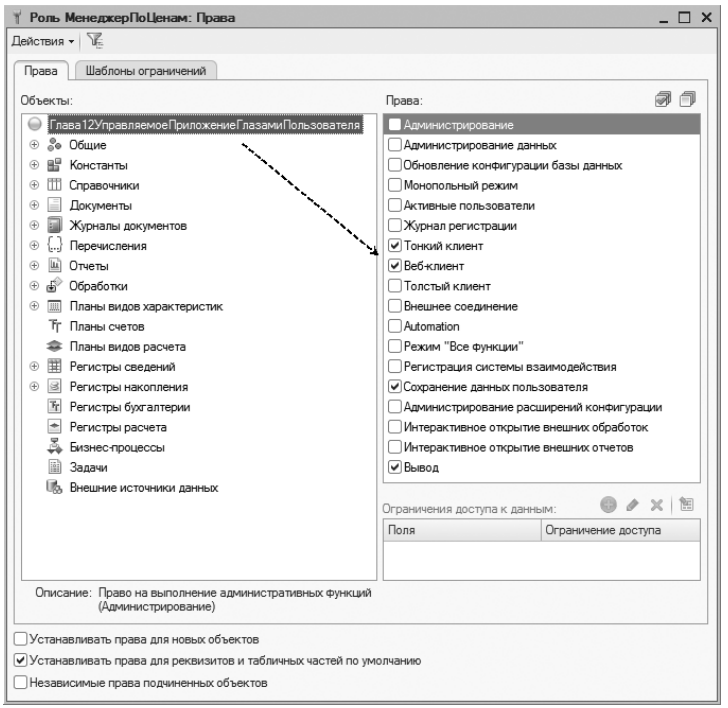


Рис. 1.54. Право на конфигурацию у роли «МенеджерПоЦенам»

То есть у новой роли есть право на запуск тонкого и веб-клиента, на сохранение данных и вывод информации. Нас это устраивает.

Как и для других объектов конфигурации, для подсистем определены права. Доступность раздела в командном интерфейсе настраивается для роли установкой права Просмотр подсистемы, соответствующей разделу.

Установим право Просмотр для подсистемы Ценообразование (рис. 1.55). Установленное право обеспечит доступность раздела Ценообразование в командном интерфейсе пользователя с ролью Менеджер по ценам.

Для справочника Товары установим право Просмотр – пользователь должен иметь возможность просматривать и выбирать товары, для которых он будет устанавливать/изменять цены (рис. 1.56).

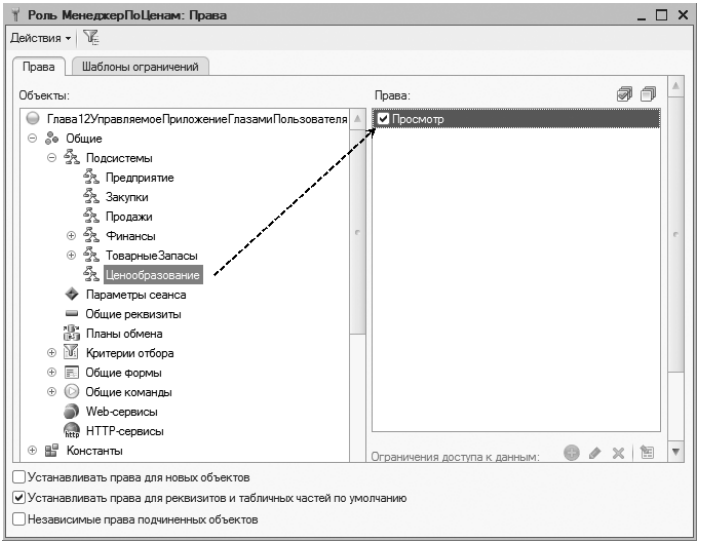


Рис. 1.55. Установка права «Просмотр» для подсистемы «Ценообразование» у роли «Менеджер по ценам»

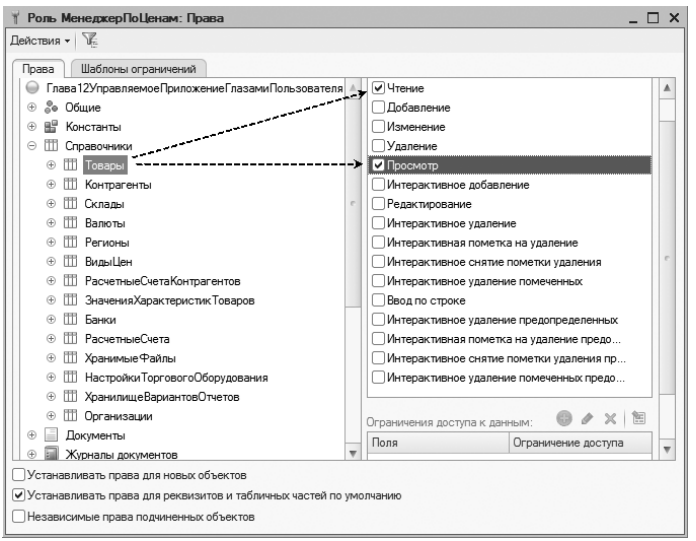


Рис. 1.56. Установка права «Просмотр» для справочника «Товары» у роли «Менеджер по ценам»

Кроме того, чтобы пользователь с ролью Менеджер по ценам мог просматривать список товаров и открывать форму просмотра товара, нужно дать роли Менеджер по ценам еще некоторые дополнительные права:

При установке интерактивного права Просмотр автоматически устанавливается основное право Чтение.

- право Чтение для справочника Организации,
- право Чтение для плана видов характеристик Виды характеристик товаров,
- право Чтение для регистра сведений Характеристики товаров,
- право Просмотр для справочника Значения характеристик товаров,
- право Просмотр для справочника Хранимые файлы.

Для справочника Виды цен установим все права (кроме прав интерактивного удаления): пользователь должен иметь полный доступ к этому справочнику для формирования используемых видов цен (рис. 1.57).

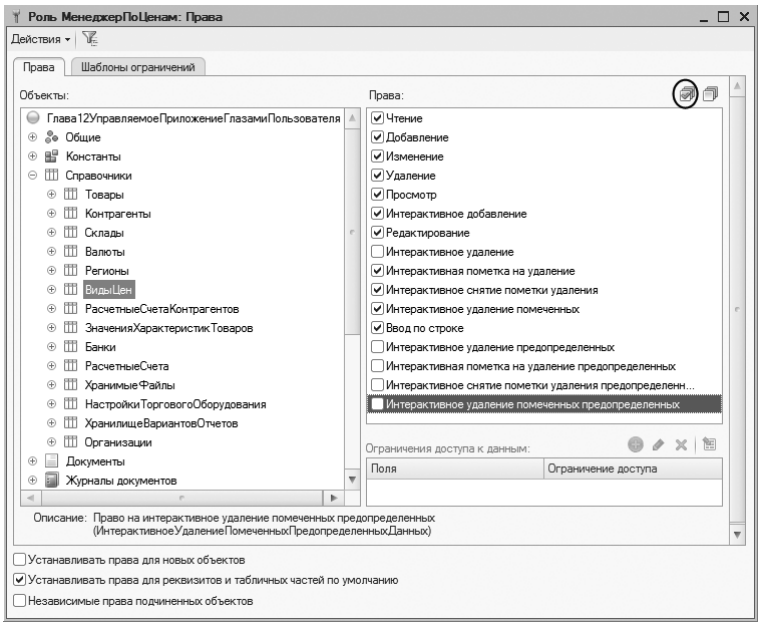


Рис. 1.57. Установка всех прав для справочника «Виды цен» у роли «Менеджер по ценам»

Для этого можно с помощью кнопки в правом верхнем углу над списком прав отметить все права и затем снять права, разрешающие интерактивное удаление элементов справочника.

Для регистра сведений Цены товаров также установим все права: пользователь должен иметь полный доступ к этому регистру для управления ценами в разрезе товаров и видов цен (рис. 1.58).

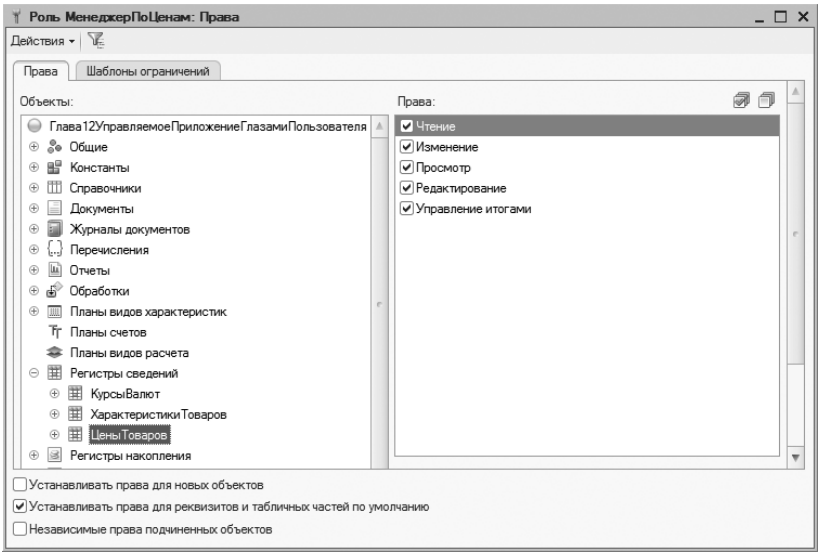


Рис. 1.58. Установка всех прав для регистра сведений «Цены товаров» у роли «Менеджер по ценам»

Перед созданием пользователя необходимо обновить конфигурацию базы данных. Это можно сделать, выбрав команду Конфигурация – Обновить конфигурацию базы данных.

Мы определили набор прав для пользователей, выполняющих операции по ценообразованию. Теперь нам необходимо создать пользователя и назначить ему вновь созданную роль. Перейдем к решению этой задачи.

Система управления пользователями

Для регистрации пользователей и назначения для них ролей используется система управления пользователями. Для знакомства с этой системой добавим в наше приложение нового пользователя, который будет осуществлять управление ценами.

Командой Администрирование – Пользователи откроем список пользователей и воспользуемся командой Добавить контекстного меню этого списка (рис. 1.59).

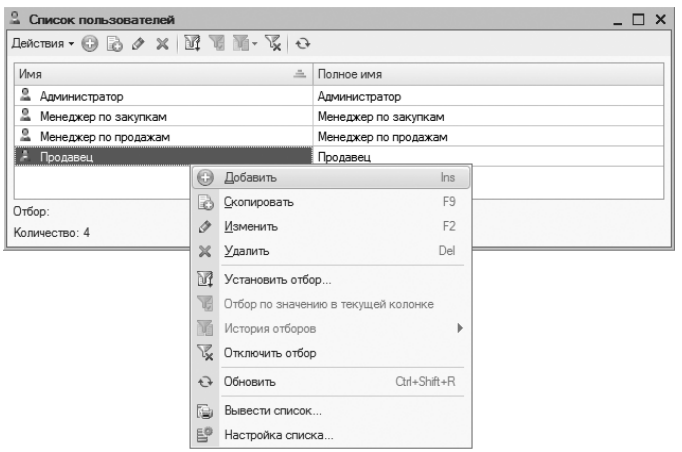


Рис. 1.59. Добавление нового пользователя

В открывшемся окне определим значения свойств учетной записи добавляемого пользователя (рис. 1.60).

На закладке Основные заполним свойства:

- Имя – краткое имя пользователя, которое вводится в диалоге аутентификации пользователя (должно быть уникальным среди всех пользователей данного прикладного решения); задаем имя «Менеджер по ценам».

На закладке Прочие заполним свойства:

- Доступные роли – указываются те роли, которые есть у пользователя при работе с прикладным решением; помечаем роль «Менеджер по ценам».

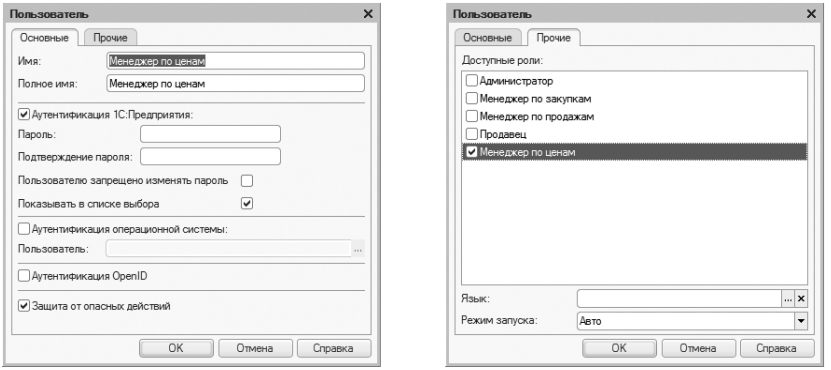


Рис. 1.60. Назначение пользователю имени и роли

Права складываются по правилам логического или – если хотя бы в одной из доступных ролей право на выполнение операции установлено, то пользователь может выполнить операцию.

Сохраним сведения о пользователе, нажав кнопку OK, обновим конфигурацию базы данных и запустим наше приложение от имени созданного пользователя (рис. 1.61).

Для менеджера по ценам доступен раздел глобального командного интерфейса Ценообразование и команды этого раздела.

Выберем команду Товары для открытия списка товаров – в рабочей области открылась форма списка справочника. Развернем группу товаров Обувь.

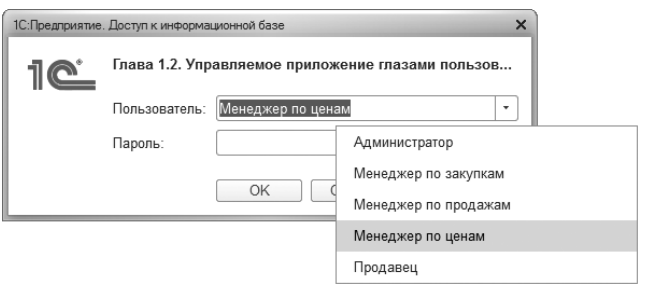


Рис. 1.61. Запуск приложения от имени пользователя «Менеджер по ценам»

В колонке Поставщик вместо данных присутствует сообщение Объект не найден... (рис. 1.62).

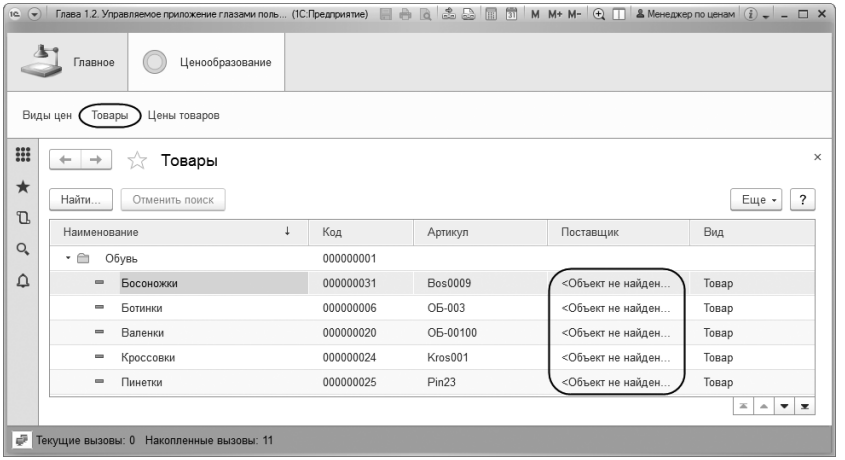


Рис. 1.62. Для пользователя «Менеджер по ценам» справочник «Товары» доступен, а справочник «Контрагенты» недоступен

Все дело в том, что для справочника Контрагенты у роли Менеджер по ценам все права сброшены – для вновь создаваемой роли права на существующие объекты по умолчанию не устанавливаются.

Чтобы в списке отображался поставщик, достаточно установить у роли основное право Чтение для справочника Контрагенты (рис. 1.63).

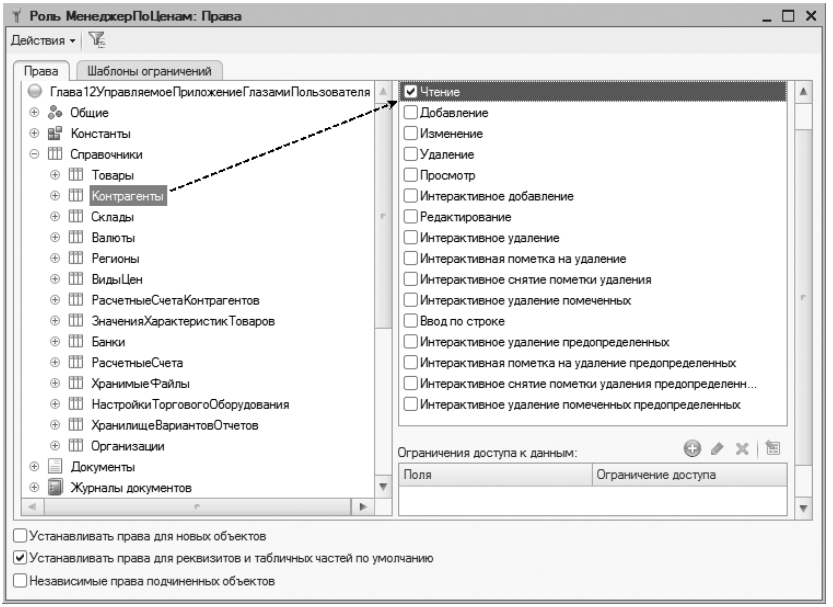


Рис. 1.63. Разрешение пользователю «Менеджер по ценам» читать справочник «Контрагенты»

Теперь при открытии списка товаров в колонке Поставщик отображается наименование поставщика (рис. 1.64).

Пойдем дальше и откроем форму товара. Пользователь, управляющий ценами, должен иметь возможность распечатать штрихкод товара. Однако в открытой форме нет команды, позволяющей выполнить эту операцию (рис. 1.65).

Непосредственно печать штрихкода выполняет команда Печать штрихкода, подчиненная справочнику Товары. А использовать эту команду для роли запрещено – право Просмотр не установлено.

ВНИМАНИЕ!

При установке прав на объект конфигурации права на его подчиненные команды автоматически не устанавливаются.

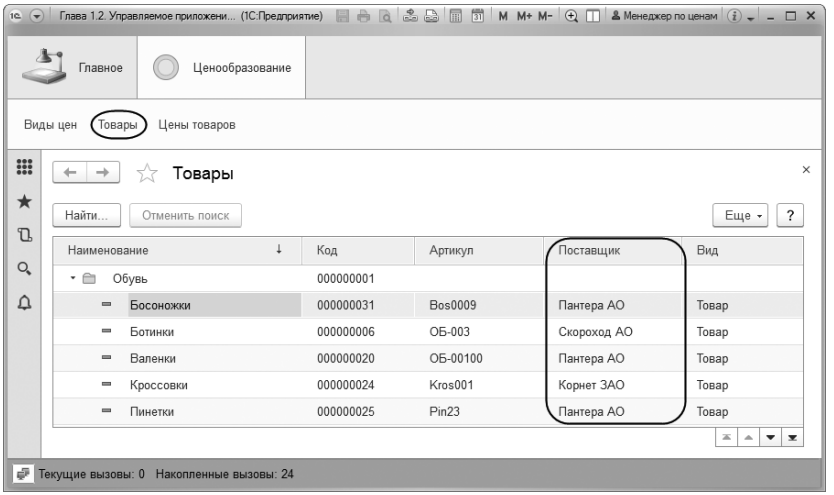


Рис. 1.64. Для пользователя «Менеджер по ценам» доступны справочники «Товары» и «Контрагенты»

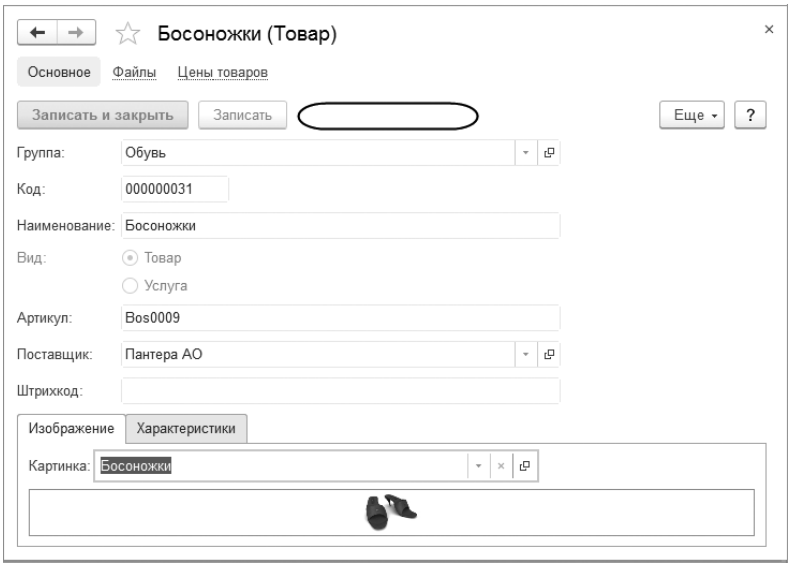


Рис. 1.65. В форме товара отсутствует команда печати штрихкода

Исправим ситуацию установкой права Просмотр для подчиненной команды (рис. 1.66).

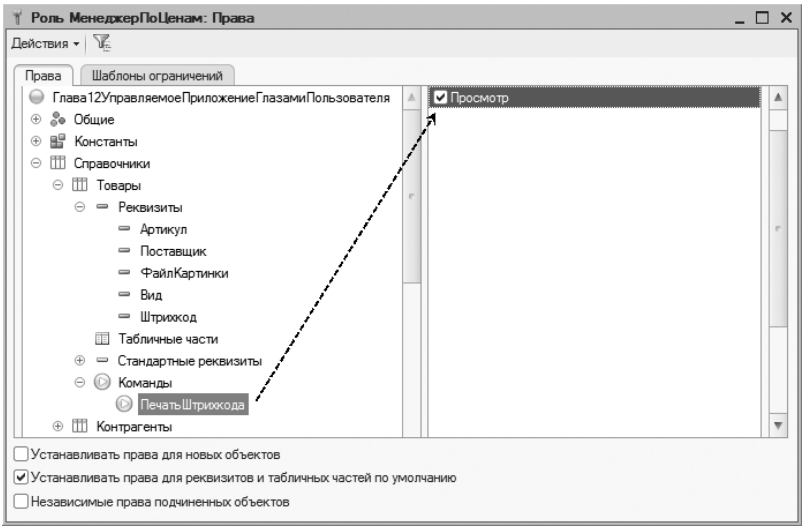


Рис. 1.66. Разрешение пользователю «Менеджер по ценам» использовать подчиненную команду «Печать штрихкода»

Сохраним изменения, запустим демонстрационную базу в режиме 1С:Предприятие от имени пользователя Менеджер по ценам и опять откроем форму товара. Теперь за счет установленного права в ней присутствует команда Печать штрихкода (рис. 1.67).

Все, мы полностью закончили настройку прав для роли Менеджер по продажам.

На текущий момент для управления ценами мы обеспечили формирование отдельного раздела командного интерфейса, формирование набора команд для этого раздела и ролевую настройку прав пользователей, занимающихся ценообразованием.

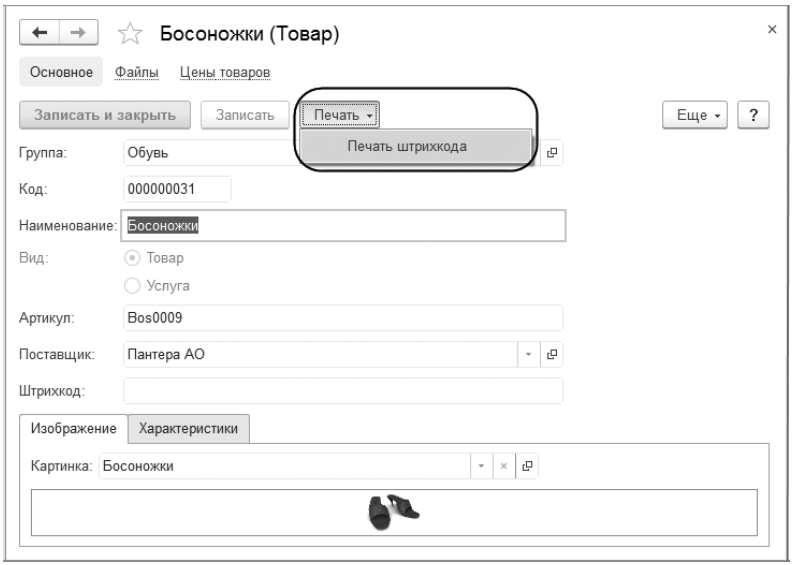


Рис. 1.67. В форме товара присутствует команда печати штрихкода

Глава 1.6. Редактирование командного интерфейса

Одним из условий комфортной работы пользователя с командным интерфейсом является «минимализм» последнего – в идеале пользователь должен иметь удобный доступ только к тем командам, которые необходимы для решения его задач.

В предыдущей главе мы рассмотрели механизм ролевой настройки *доступности* команд в командном интерфейсе. Этот механизм позволяет эффективно добиться выполнения условия минимализации командного интерфейса – доступны только те команды, которые необходимы пользователю.

С другой стороны, хотелось бы обеспечить пользователю максимально комфортные условия для работы. Ведь среди доступных команд есть такие, к которым пользователи обращаются очень часто, и такие, доступ к которым необходим изредка. Хорошо бы иметь возможность не отображать в командном интерфейсе вторую группу команд, оставив пользователю возможность обращаться к ним при необходимости.

Для решения этой задачи предназначен *механизм настройки размещения и видимости команд по ролям*. Он позволяет оптимальным образом настроить командный интерфейс для различных ролей пользователей – показать часто используемые команды и скрыть те, которые используются редко.

ВНИМАНИЕ!

Видимость команды вступает в силу только тогда, когда команда доступна. Недоступные для роли команды не попадут в командный интерфейс независимо от настройки видимости.

При редактировании размещения и видимости команд можно выделить три уровня настройки:

- Не настраиваем – система автоматически разместит команды в командном интерфейсе и настроит их видимость.

- Настраивается разработчиком – в режиме конфигурирования настраиваются размещение и видимость команд в разрезе ролей. Эта настройка будет настройкой по умолчанию, используемой для всех пользователей с определенной ролью.
- Настраивается пользователем – в режиме эксплуатации каждый пользователь настраивает видимость команд исходя из собственных предпочтений. При этом пользователи с одной и той же ролью могут определить различный состав видимых по умолчанию команд.

Возможности настройки командного интерфейса пользователем будут рассмотрены в главе 1.8 на стр. 126. Сейчас же мы рассмотрим первые два уровня.

В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.5. Настраиваем доступность команд по ролям». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.6. Редактирование командного интерфейса».

Автоматическое размещение и видимость команд

На данный момент пользователю с ролью Менеджер по ценам в панели функций раздела Ценообразование видимы команды открытия форм списков. Другие команды, как мы видим, отсутствуют – поэтому группы команд в панели функций не отображаются (рис. 1.68).

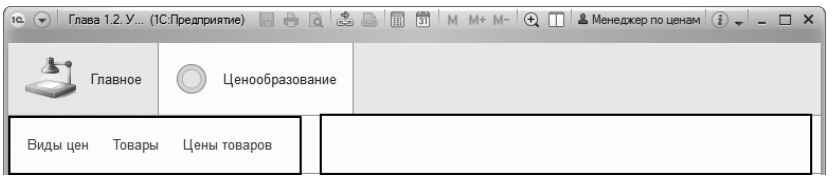


Рис. 1.68. Автоматически сформированный набор видимых команд раздела «Ценообразование»

А где же команды создания новых объектов? Эти команды «спрятаны». Мы не вмешивались в процесс размещения и установки видимости команд. Система разместила команды командного интерфейса и установила их видимость автоматически.

Правила, которым следует система при размещении команд командного интерфейса, описаны в разделе «Правила размещения глобальных команд» на стр. 93. Правила автоматической настройки видимости команд приведены в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 6.2.7.

Такая автоматическая видимость команд объясняется достаточно просто. В общем случае просмотр списков требуется многим пользователям, а вот операция создания новых элементов выполняется реже. Поэтому проще настроить видимость команд создания элементов для нескольких пользователей, чем настраивать видимость команд открытия списков для многих пользователей.

Как же создавать новые объекты (элементы справочников, документы и т. д.), когда пользователь не видит соответствующих команд? В этом случае можно воспользоваться командами, предоставляемыми формами списков.

Например, новый вид цен можно создать из формы списка справочника Виды цен (рис. 1.69).

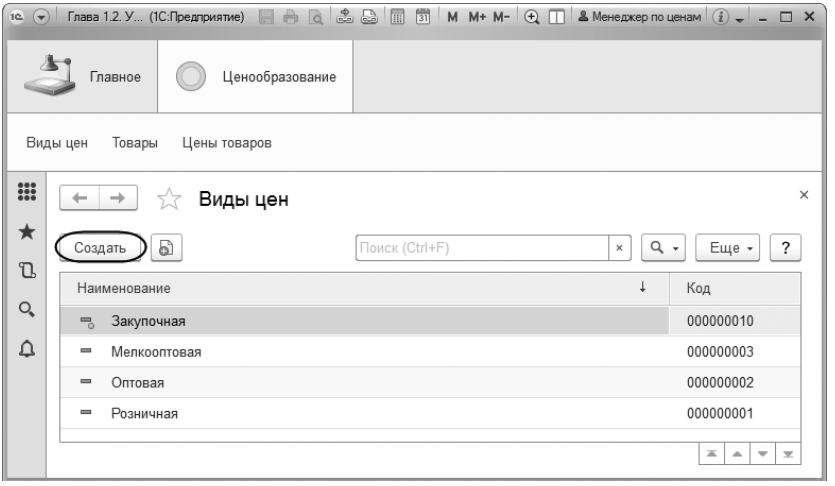


Рис. 1.69. Работа с видами цен из формы списка

При таком доступе к команде создания нового объекта пользователь должен выполнить дополнительное действие – открыть форму списка. Для справочника Виды цен это достаточно удобно, так как состав видов цен изменяется редко и, соответственно, пользователь редко обращается к команде создания нового элемента.

А вот команда назначения цены на товар нужна часто, и выполнение дополнительного действия может быть неудобно. Для обеспечения более комфортного доступа к команде ее необходимо сделать видимой в командном интерфейсе.

Еще одной командой, видимость которой для различных пользователей нужно настроить, является команда Товары, открывающая список товаров для просмотра.

Кроме видимости следует обратить внимание и на порядок команд. Сейчас команды в панели навигации расположены не очень удачно. Самой важной для пользователя является команда открытия списка цен на товары, а она расположена в самом конце списка. Хотелось бы, чтобы эта команда отображалась первой.

Для решения подобных задач система «1С:Предприятие» предоставляет редакторы, позволяющие настраивать размещение и видимость команд в командном интерфейсе в соответствии с требованиями прикладного решения.

ВНИМАНИЕ!

Редактирование командного интерфейса не позволит добавить или исключить команды. Эта операция позволит лишь оптимизировать командный интерфейс – обеспечить удобство работы для различных групп пользователей.

Однако, прежде чем заниматься настройкой, давайте познакомимся с автоматическим размещением и видимостью команд.

Категории и группы команд

На размещение команд командного интерфейса влияет *категория* команды. По сути, категория – это еще одна классификация команд. В ней команды разделяются по двум признакам: зависимость команды от дополнительных данных и назначение команды.

По зависимости от данных команды делятся:

- на независимые,
- параметризуемые.

По назначению команды делятся:

- на навигационные,
- команды действий.

Независимые команды не требуют для своего исполнения никаких дополнительных данных. Это значит, что результат выполнения такой команды будет одинаков независимо от того, из какого окна (или формы) вызвана команда, какие данные обрабатываются пользователем в момент вызова команды, и т. д. К независимым относятся команды открытия списка справочника, создания нового документа и др.

Например, в результате выполнения команды Товары в рабочей области основного окна приложения открывается форма списка справочника Товары (рис. 1.70). При этом дополнительная информация для открытия списка команде не требуется.

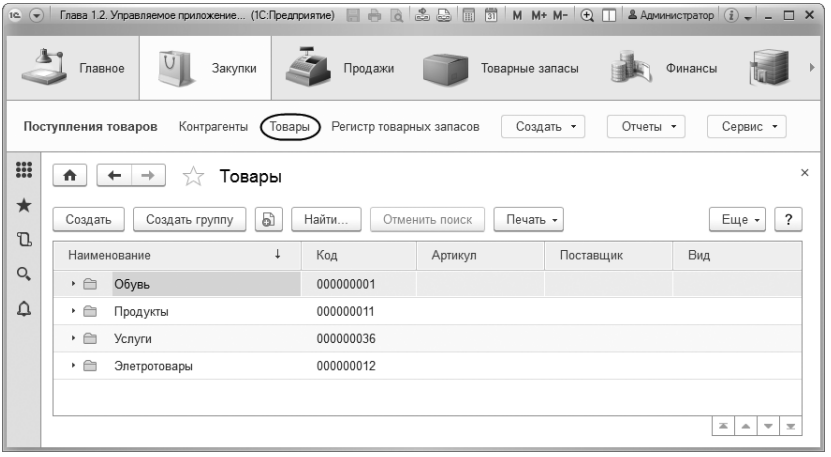


Рис. 1.70. Независимая команда не требует параметра

Параметризуемые команды для своего выполнения требуют дополнительные данные – параметры выполнения, которые определяют результат исполнения команды. К параметризуемым относятся команды открытия списка подчиненного справочника (параметр – ссылка на элемент справочника-владельца), создания нового документа на основании (параметр – объект, на основании которого создается документ) и др.

ВНИМАНИЕ!

> Источником значения для параметра могут служить только данные формы. Поэтому параметризуемые команды должны располагаться в форме. Причем в той, которая содержит данные подходящего для параметра типа.

Например, команда Файлы, вызванная из формы элемента справочника Товары, откроет (в том же окне) форму списка элементов справочника Хранимые файлы (рис. 1.71). В списке будут отображены элементы, подчиненные редактируемому элементу справочника Товары.

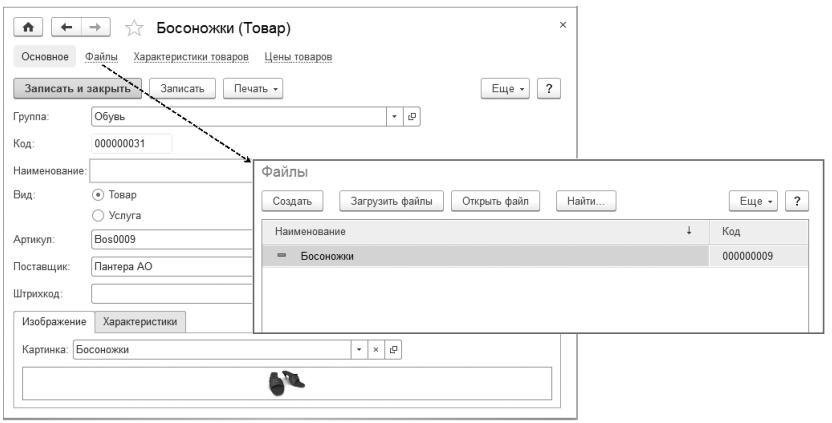


Рис. 1.71. Параметризуемая команда требует параметр

Различие между независимыми и параметризуемыми командами можно продемонстрировать на следующем примере. Предположим, нам необходимо получить отчет по остаткам товаров на складе Большой. Мы можем выбрать раздел Товарные запасы и воспользоваться независимой командой Остатки товаров (рис. 1.72).

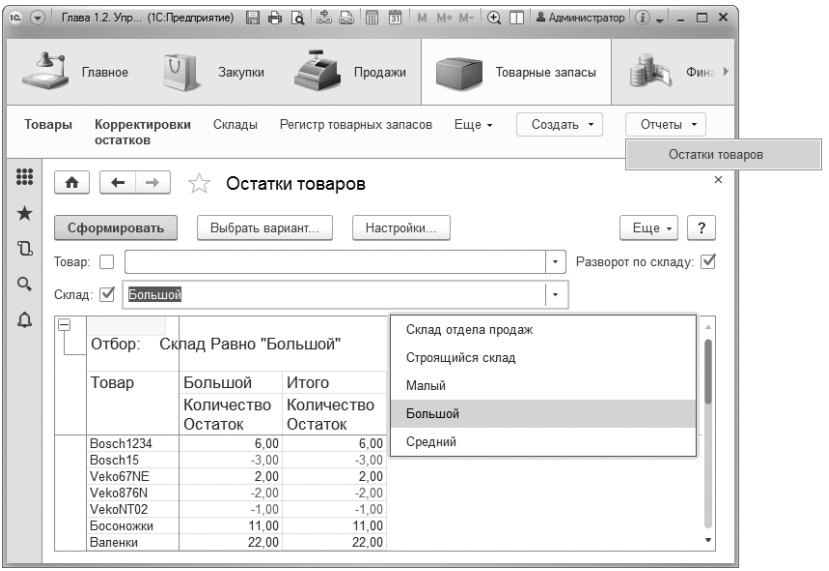


Рис. 1.72. Вызов отчета независимой командой «Остатки товаров»

Но в этом случае нам необходимо в настройках отчета выбрать интересующий нас склад. Для исключения этапа выбора склада мы можем вызвать отчет из формы списка складов. Для этого выделяем в списке интересующий нас склад и выбираем параметризуемую команду Остатки по складу (рис. 1.73).

В этом случае нет необходимости выбирать склад в настройках отчета: он передан команде как параметр.

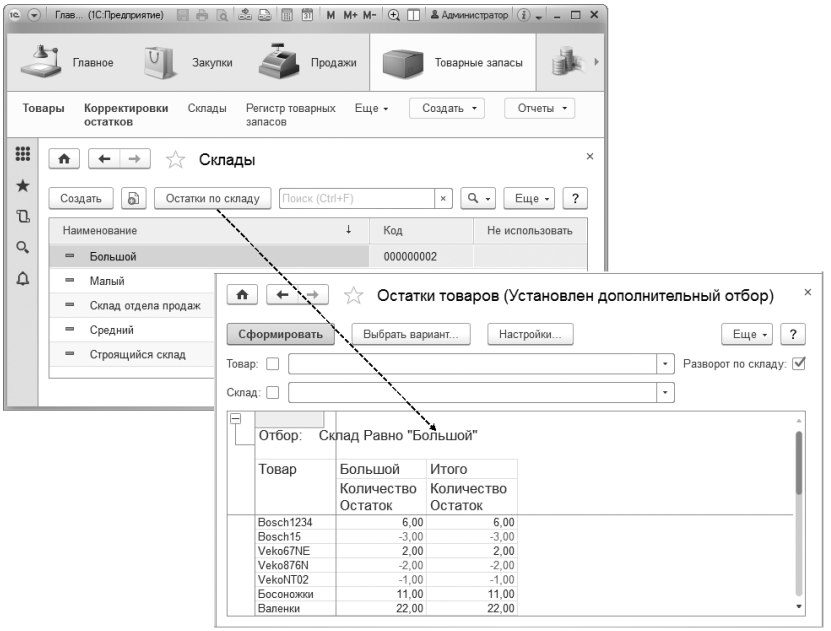


Рис. 1.73. Вызов отчета параметризуемой командой «Остатки по складу»

Навигационные команды предназначены для навигации пользователя по функциональности прикладного решения. Исполнение навигационной команды обычно приводит к открытию новой формы списка в окне, из которого была вызвана команда. Если в рабочей области окна отображалась какая-либо форма, то она замещается на новую.

Навигационными могут быть как независимые, так и параметризуемые глобальные команды. К навигационным относится команда перехода к форме списка справочника – это независимая команда; команда перехода к форме списка подчиненного справочника – это параметризуемая команда.

Например, команда Продажи, вызванная из основного окна приложения, откроет форму списка документов Продажи товара в рабочей области этого окна (рис. 1.74).

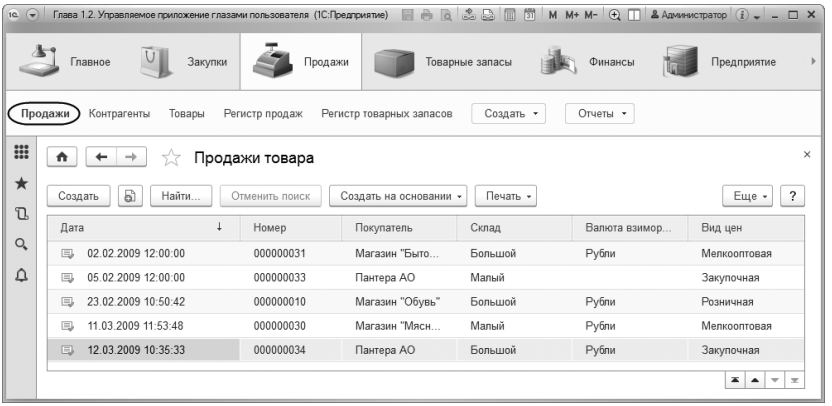


Рис. 1.74. Навигационная команда

Команды действий предназначены для непосредственного выполнения какой-либо задачи по обработке данных. К командам действий относится команда создания нового документа – независимая; команда вызова отчета об остатках по складу – параметризуемая. Например, команда Продажа в рабочей области основного окна откроет окно клиентского приложения для ввода нового документа Продажа товара (рис. 1.75).

Различие между командами навигации и командами действий можно продемонстрировать на следующем примере. Предположим, нам необходимо создать новый вид цены.

Но сначала убедимся, что такого вида цены нет. Для этого откроем список видов цен с помощью навигационной команды Виды цен из подменю Еще панели функций раздела Товарные запасы. Если нужного вида цены нет, для его создания можно воспользоваться командой действия Вид цены из подменю Создать (рис. 1.76).

Для команды создания нового вида цены предварительно была выполнена настройка видимости.

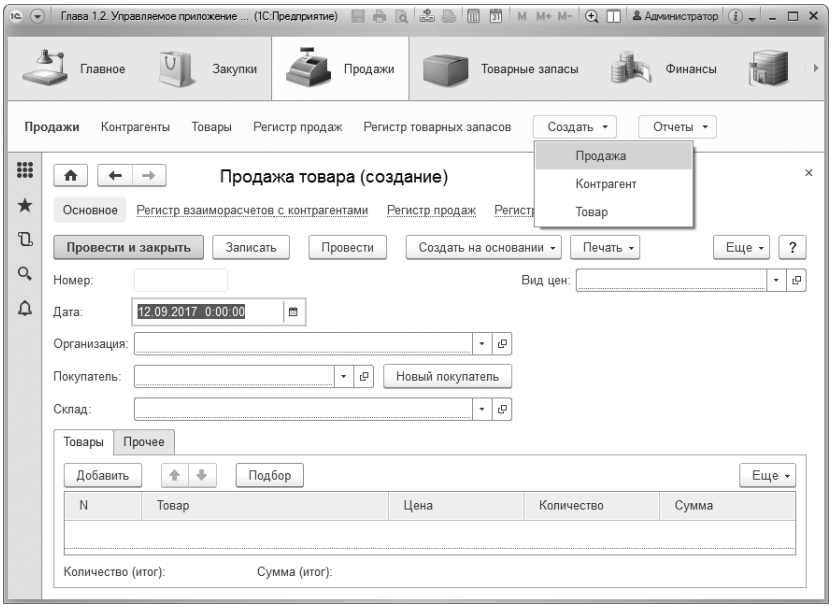


Рис. 1.75. Команды действий

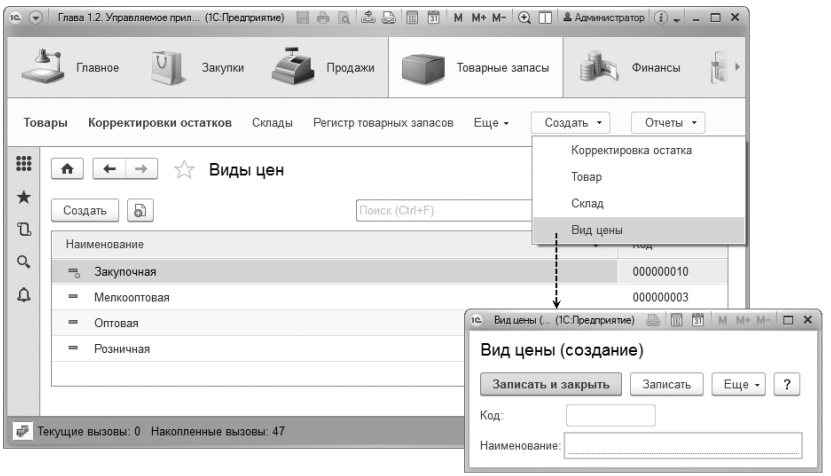


Рис. 1.76. Команда действия – для обработки данных

Однако гораздо удобнее создавать элементы данных непосредственно при вводе значений в полях ссылочного типа. Например, можно создать новый вид цены «по себестоимости», начав ввод символов в поле Вид цен документа Продажа товара. Если совпадений с новым значением не найдено, вы можете нажать кнопку создания нового элемента (с пиктограммой «+») и продолжить ввод наименования в отдельном окне для ввода нового вида цены. При этом введенные вами символы уже будут подставлены в наименование нового элемента справочника (рис. 1.77).

Чтобы возможность создания элементов справочников при вводе ссылочных значений была доступна, необходимо, чтобы у справочников или у ссылающихся на них реквизитов свойство Создание при вводе было установлено в значение Использовать.

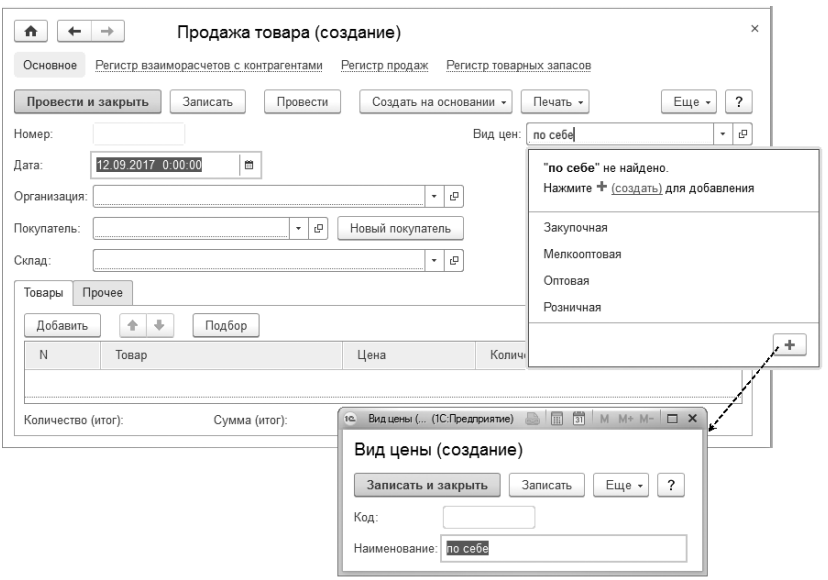


Рис. 1.77. Создание элементов данных при вводе ссылочных значений

После записи нового элемента ссылка на этот элемент автоматически подставится в поле ввода.

Правила размещения глобальных команд

В начале панели функций текущего раздела (см. рис. 1.78 вверху) или в меню функций раздела (см. рис. 1.78 внизу) основного окна приложения размещаются *независимые навигационные глобальные команды*.

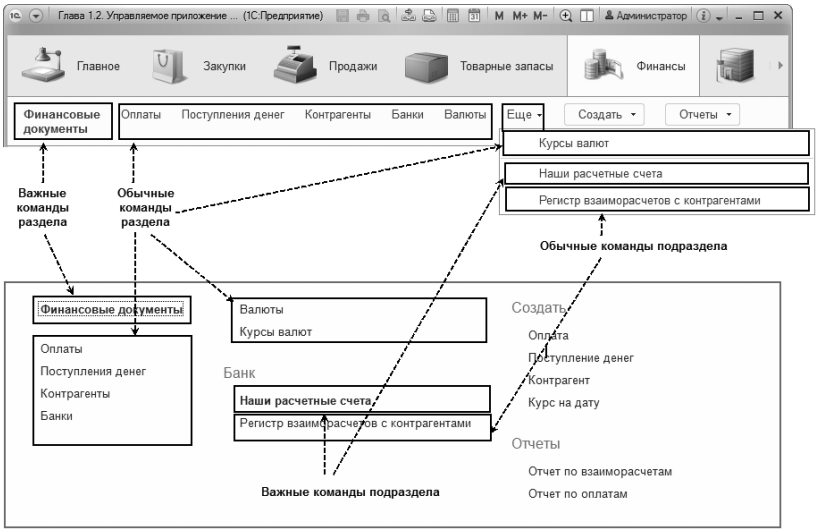


Рис. 1.78. Навигационные команды основного окна приложения

Как видно на рис. 1.78, команды раздела и подразделов показываются и в панели функций, и в меню функций, но по-разному. Из-за ограниченного пространства в панели функций используется единственный способ выделения команд – отображение их жирным шрифтом. В меню функций места достаточно, поэтому для отображения тех же самых команд там используется не только выделение жирным шрифтом, но и группировка команд в группы.

Если команда относится к подчиненной подсистеме, то в панели функций она никак не выделяется, а в меню функций она выделяется в отдельную группу с заголовком этой подсистемы. Для каждого подраздела в меню функций формируется отдельная группа. При размещении команд сначала выводятся команды раздела, а затем – блоки команд каждого из подразделов.

Если необходимо выделить какие-то команды для перехода к наиболее важной или наиболее часто используемой функциональности, то ее нужно поместить в группу Важное. Такие команды будут показаны первыми и выделены жирным шрифтом. Остальные команды будут иметь обычный приоритет и не будут никак выделяться.

За ними следуют произвольные группы команд, созданные разработчиком. Эти группы образуются объектами конфигурации ГруппаКоманд. Произвольные группы в меню функций раздела имеют собственный заголовок и выделяются в отдельные группы, а в панели функций никак не выделяются.

Если команды выбранного раздела отображаются в панели функций, то, в случае если все команды не помещаются в панели, на панели отображается подменю Еще, которое содержит оставшиеся навигационные команды.

После навигационных команд располагаются *независимые глобальные команды действий*, собранные в группы (рис. 1.79).

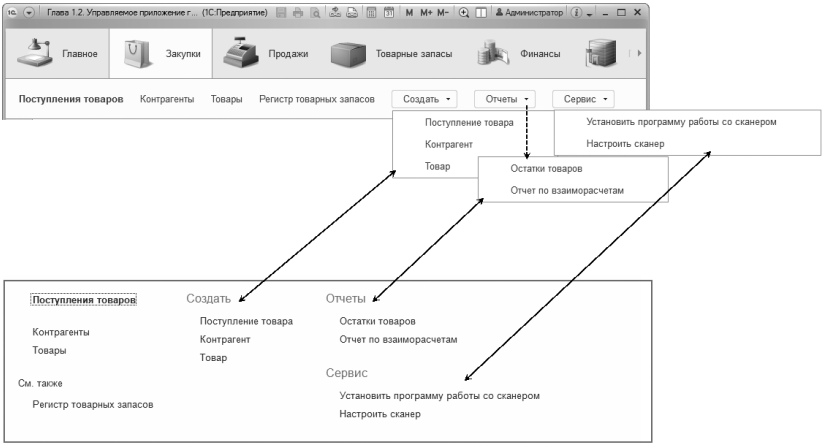


Рис. 1.79. Команды действий основного окна приложения

Команды действий могут размещаться в одной из трех стандартных групп, которые отображаются в меню функций в виде подменю:

- Создать – команды открытия окон с формами ввода данных;

- Отчеты – команды открытия окон с формами отчетов;
- Сервис – команды открытия окон с формами обработок.

Команды также могут быть размещены и в произвольных группах, созданных разработчиком. Эти группы располагаются после стандартных и представлены синонимом соответствующего объекта конфигурации `ГруппаКоманд`.

Команды действий по подразделам не структурируются. В группах объединены команды, принадлежащие как подсистеме верхнего уровня иерархии, так и всем подчиненным подсистемам.

В панели навигации окна клиентского приложения размещаются *параметризуемые навигационные глобальные команды*. С использованием этих команд осуществляется навигация по данным, логически связанным с данными, обрабатываемыми в основной форме. Например, команда перехода к записям регистра, подчиненным редактируемому документу (рис. 1.80).

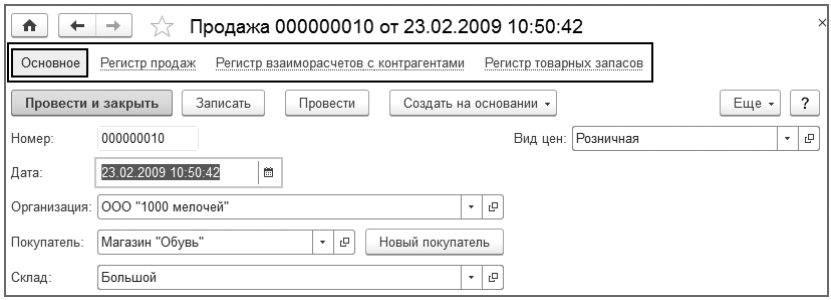


Рис. 1.80. Команды панели навигации окна клиентского приложения

В панели навигации окна клиентского приложения первой всегда является команда Основное, которая предназначена для возврата к основной форме окна приложения.

Остальные команды предназначаются для открытия вспомогательных форм данных, связанных с данными, которые отображаются в основной форме.

После стандартных команд могут быть размещены произвольные группы команд, созданные разработчиком. Эти группы не имеют собственных заголовков в панели навигации и никак не выделяются.

Если все команды не помещаются в панели навигации формы, то в самом конце панели отображается подменю Еще, которое содержит оставшиеся команды.

В командной панели основной формы окна клиентского приложения размещаются *параметризуемые глобальные команды действий*. С использованием этих команд выполняется обработка данных, отображаемых в форме. Например, команда проведения документа (рис. 1.81).

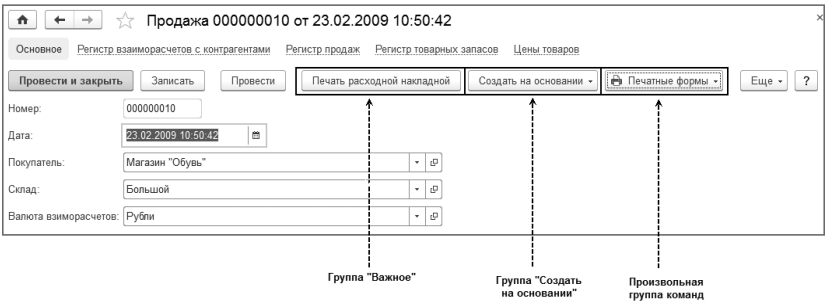


Рис. 1.81. Команды командной панели окна клиентского приложения

Команды, источником которых является основной реквизит формы, всегда располагаются в самом начале командной панели.

Затем идут команды из группы Важное, каждая из которых отображается своей кнопкой на панели. Затем идет группа команд Создать на основании, которая показывается в виде подменю.

Также в виде подменю отображаются все группы команд, созданные разработчиком. Эти группы располагаются после стандартных и представлены синонимом соответствующего объекта конфигурации ГруппаКоманд.

Если в какой-либо группе нет ни одной команды, то группа не показывается. Если все команды не помещаются в командной панели формы, то в самом конце панели отображается подменю Еще, которое

содержит оставшиеся команды. Правее может располагаться только форма вызова справки, если для формы установлена справочная информация.

Разобравшись с автоматическим размещением команд, перейдем к знакомству со средствами настройки размещения и видимости команд.

Система настройки командного интерфейса

Для настройки командного интерфейса платформа предоставляет специализированные редакторы и обеспечивает сохранение выполненных настроек.

Для хранения настроек командного интерфейса конфигурации используется свойство Командный интерфейс корневого объекта конфигурации (рис. 1.82).

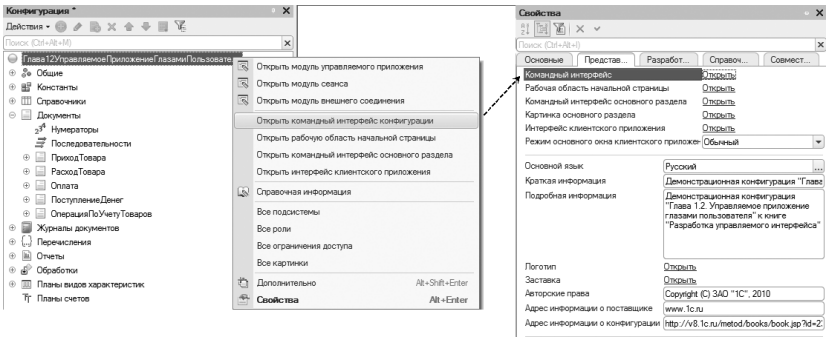


Рис. 1.82. Свойство конфигурации «Командный интерфейс»

Открыть командный интерфейс конфигурации можно из контекстного меню корня конфигурации или из палитры свойств.

В этом свойстве сохраняются:

- настройка порядка следования разделов;
- настройка видимости разделов, в том числе в разрезе ролей.

Для хранения настроек командного интерфейса раздела используется свойство Командный интерфейс соответствующей подсистемы (рис. 1.83).

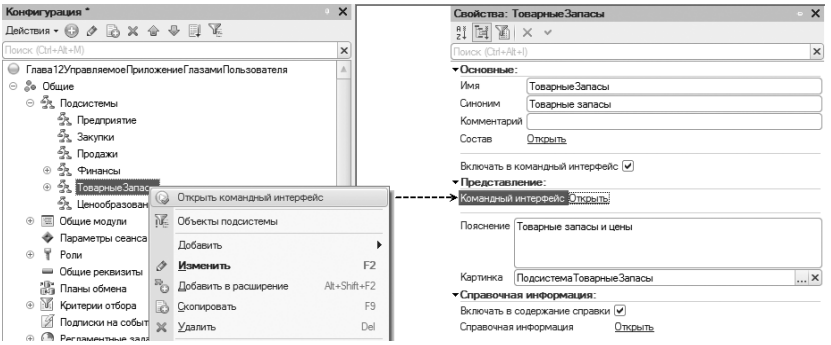


Рис. 1.83. Свойство подсистемы «Командный интерфейс»

Открыть командный интерфейс раздела можно из контекстного меню подсистемы, или из палитры свойств, или из окна редактирования объекта конфигурации Подсистема.

В этом свойстве сохраняются:

- настройка расположения команд в группах команд;
- настройка порядка следования команд;
- настройка видимости команд, в том числе в разрезе ролей.

Хранение настроек в свойстве подсистемы позволяет независимо настраивать каждый из разделов (подразделов) командного интерфейса.

Настройки основного раздела сохраняются в свойстве Командный интерфейс основного раздела корневого объекта конфигурации (рис. 1.84).

Открыть командный интерфейс основного раздела можно из контекстного меню корня конфигурации или из палитры свойств.

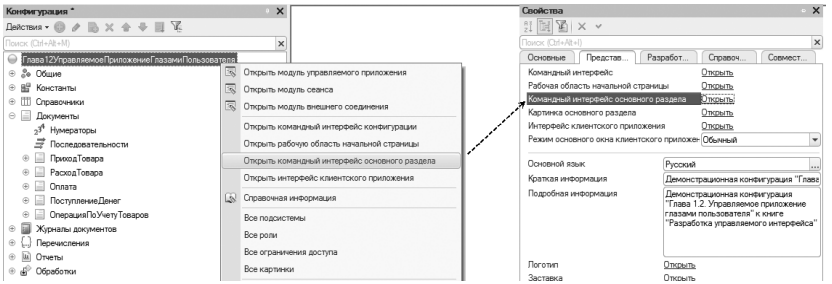


Рис. 1.84. Свойство конфигурации «Командный интерфейс основного раздела»

Для редактирования командного интерфейса система предоставляет специализированные инструменты:

- редактор командного интерфейса,
- редактор командного интерфейса конфигурации,
- редактор командного интерфейса основного раздела,
- редактор «Все подсистемы».

Основные приемы работы во всех редакторах совпадают. Мы их подробно рассмотрим применительно к редактору командного интерфейса.

Редактор командного интерфейса

Редактор командного интерфейса предназначен для редактирования командного интерфейса отдельного раздела (подсистемы).

Для открытия редактора можно нажать кнопку Командный интерфейс в окне редактирования свойств подсистемы либо из контекстного меню подсистемы выполнить команду Открыть командный интерфейс. В результате откроется окно редактора, в котором отображается дерево команд, доступных в той подсистеме, для которой вызван редактор (рис. 1.85).

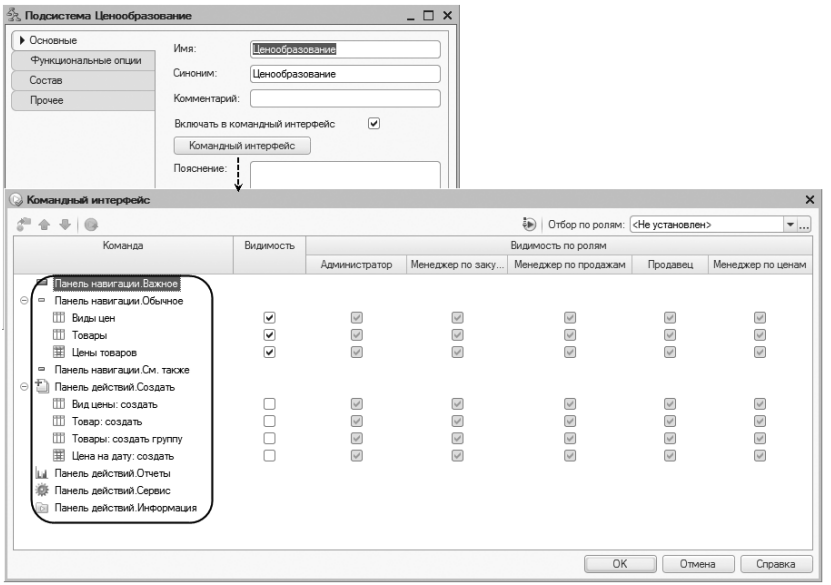


Рис. 1.85. Вызов редактора командного интерфейса подсистемы

Для *настройки размещения* команд в глобальном командном интерфейсе редактор предоставляет следующие возможности:

- можно изменить порядок следования команд внутри группы;
- можно изменить группу, в которой будет отображаться команда.

Порядок следования команд внутри группы можно изменять командами Переместить вверх и Переместить вниз (вызвав их из контекстного меню или из командной панели) или перетаскивая команду на нужную позицию мышью (рис. 1.86).

Группу, в которой будет отображаться команда, можно изменять командой Переместить команду (вызвав ее из контекстного меню или из командной панели) или перетаскивая команду в нужную группу мышью (рис. 1.87).

Переместить команду в группу, принадлежащую другой панели (имеющую другую категорию), редактор не позволит.

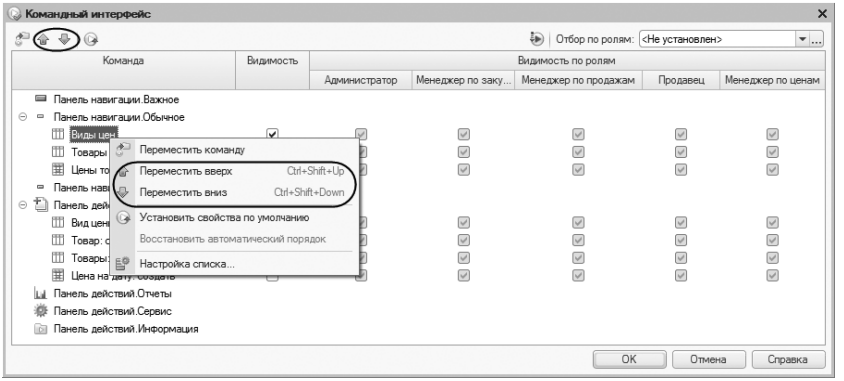


Рис. 1.86. Настройка порядка следования команд

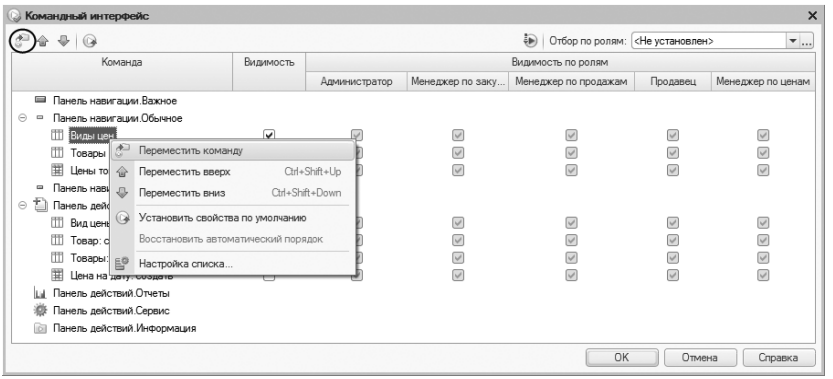


Рис. 1.87. Настройка принадлежности команды к группе

При установке порядка следования команд, отличающегося от автоматического, рядом с группой будет отображаться текст Ручной порядок (рис. 1.88).

Например, перетащили команду Товары в первую позицию группы ПанельНавигации.Обычное (см. рис. 1.88). В результате рядом с этой группой появился текст Ручной порядок.

Если необходимо вернуться к автоматическому порядку следования команд внутри группы, то это можно сделать командой Восстановить автоматический порядок команд из контекстного меню группы (рис. 1.89).

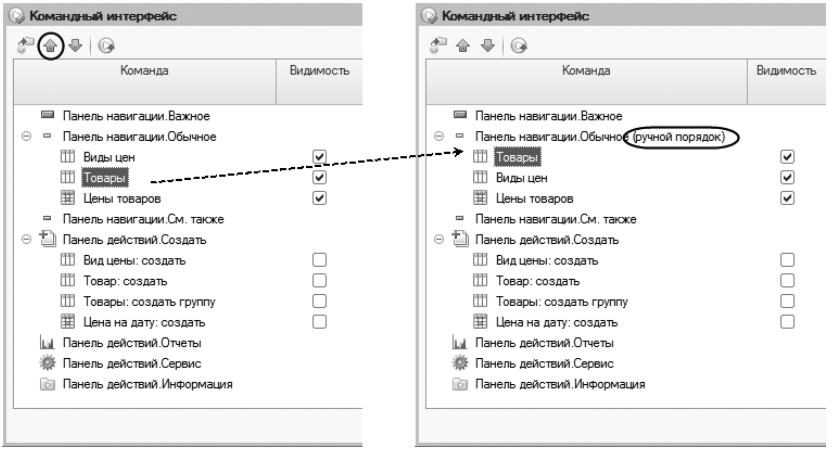


Рис. 1.88. Признак ручного изменения порядка следования команд

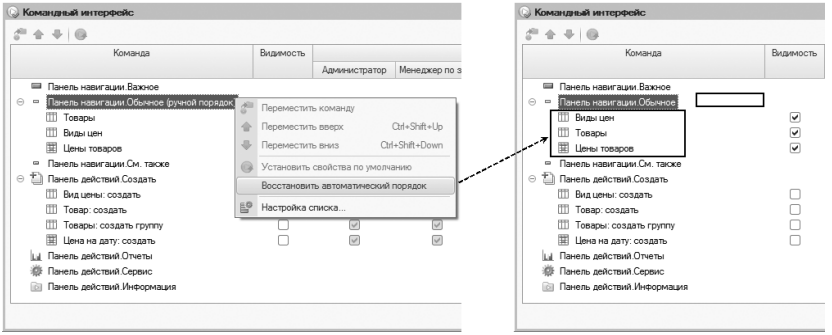


Рис. 1.89. Восстановление автоматического размещения команд

Например, для группы ПанельНавигации.Обычное выполнено восстановление автоматического порядка команд (рис. 1.89). В результате в этой группе команды упорядочились по алфавиту, а надпись «Ручной порядок» исчезла.

Для *настройки видимости по умолчанию* команд в командном интерфейсе редактор предоставляет следующие возможности:

- можно настроить общую видимость команды,
- можно настроить видимость команды в разрезе ролей.

Общая видимость команды используется для тех ролей, у которых не выполняется индивидуальная настройка видимости этой команды. Для изменения общей видимости команды необходимо установить или снять флажок в колонке Видимость для требуемой команды (рис. 1.90).

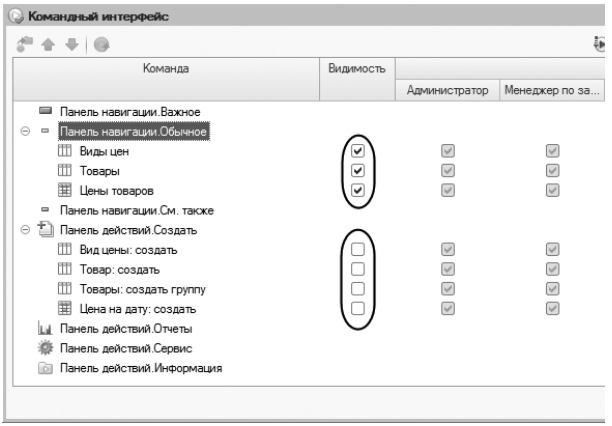


Рис. 1.90. Настройка общей видимости команд

Для общей видимости может быть задано только два значения:

- команда видима – флажок установлен;
- команда невидима – флажок сброшен.

При необходимости редактор позволяет задать *видимость по ролям* – отдельно для каждой роли, определенной в конфигурации. Для изменения видимости команды для роли необходимо установить или снять флажок в соответствующей колонке группы Видимость по ролям (рис. 1.91).

Для видимости по ролям может быть задано три значения:

- команда видима – белый фон, флажок установлен;
- команда невидима – белый фон, флажок сброшен;
- использовать общую видимость – серый фон, флажок установлен.

Значения устанавливаются щелчком левой клавиши мыши и перебираются последовательно в цикле.

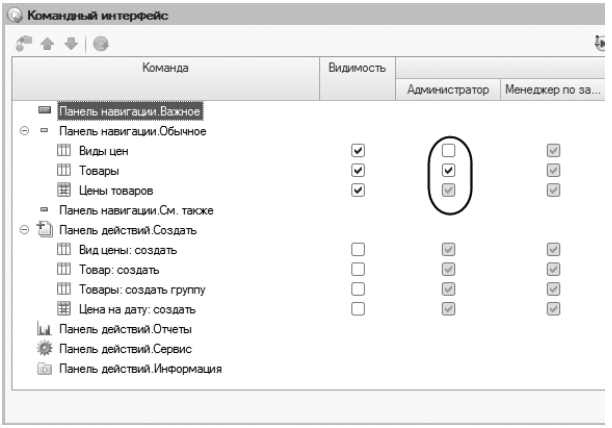


Рис. 1.91. Настройка видимости команд по ролям

В командном интерфейсе пользователя команда будет видима, если хотя бы для одной из ролей, назначенных этому пользователю, видимость для данной команды установлена.

При необходимости вернуться к автоматическим размещению и видимости команд можно воспользоваться командой контекстного меню Установить свойства по умолчанию или одноименной кнопкой командной панели (рис. 1.92).

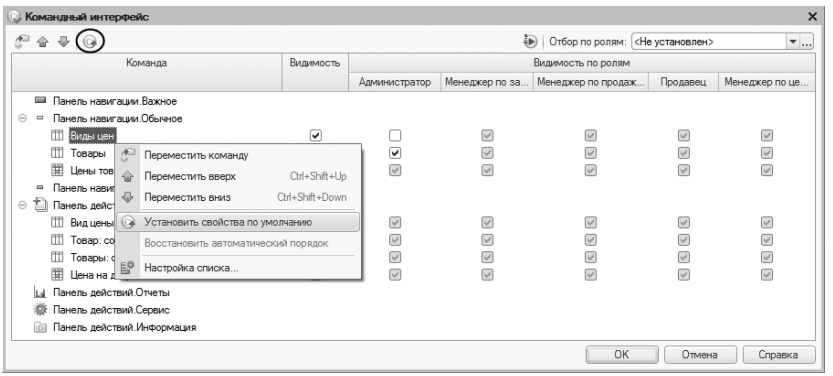


Рис. 1.92. Установка свойств команды по умолчанию

Редактор командного интерфейса поддерживает групповую настройку размещения с помощью множественного выделения команд в списке.

Для выделения диапазона команд необходимо выделить первую команду из диапазона, а затем, удерживая нажатой клавишу Shift, выделить последнюю команду из диапазона. В результате будут выбраны все команды, расположенные между первой и последней (включая граничные).

Для выделения нескольких команд, следующих не подряд, необходимо выделить первую команду, а затем, удерживая нажатой клавишу Ctrl, отмечать необходимые команды. В результате будут выделены все отмеченные команды.

С группой команд можно выполнять те же операции, что и с отдельной командой.

При необходимости разработчик может настроить состав отображаемых команд и колонки списка, в котором команды отображаются.

Для того чтобы в списке команд отобразить только видимые по умолчанию команды, следует нажать кнопку командной панели Скрыть невидимые по умолчанию (рис. 1.93). При этом также будут скрыты группы без команд.

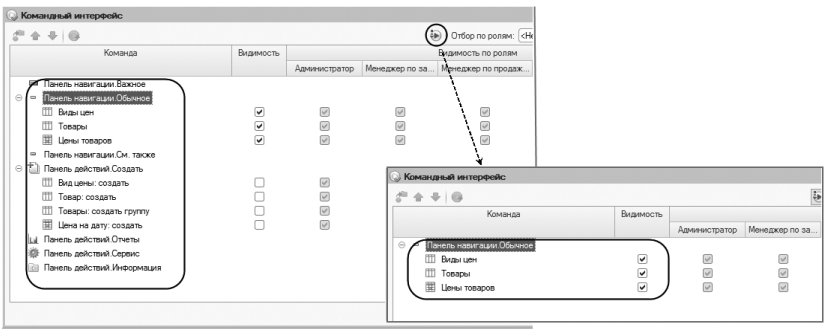


Рис. 1.93. Установка фильтра на видимые команды

Для того чтобы в списке команд отобразить только команды, доступные некоторым ролям (возможно, одной), следует задать эти роли в поле Отбор по ролям (рис. 1.94).

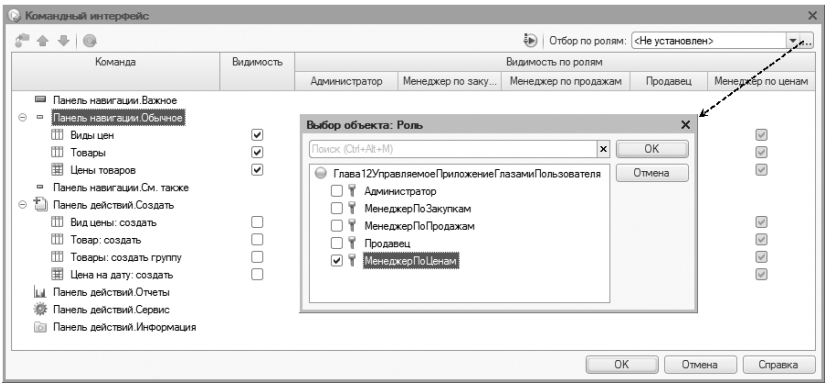


Рис. 1.94. Установка фильтра на команды, доступные роли

В списке будут отображаться только те команды, которые доступны хотя бы одной из выбранных ролей (рис. 1.95).

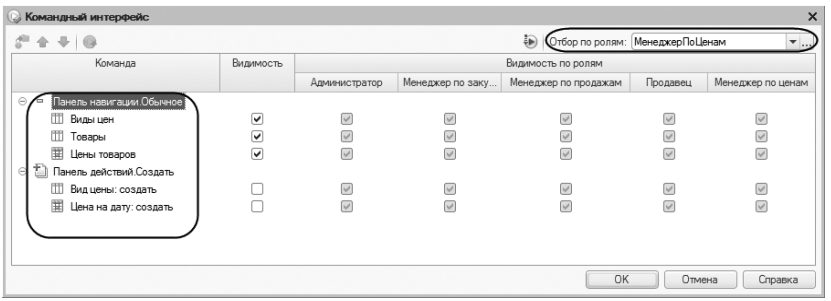


Рис. 1.95. Команды, доступные роли «Менеджер по ценам»

Например, на рис. 1.95 показаны команды Вид цены: создать и Цена на дату: создать, которые доступны роли Менеджер по ценам, но невидимы.

Если при отборе по ролям установлен отбор только видимых команд, то будут отображены только те команды, которые по умолчанию видимы хотя бы одной роли.

Для отключения отбора по ролям в поле отбора нужно выбрать значение Не установлен. Выпадающий список позволяет быстро включать один из нескольких последних установленных отборов (рис. 1.96).

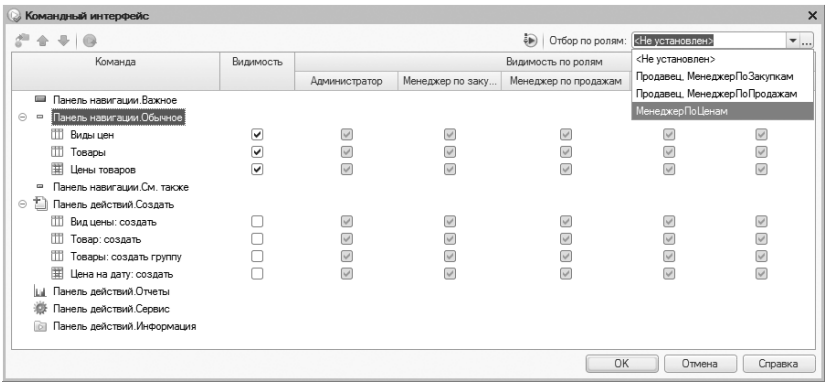


Рис. 1.96. История отбора и отключение отбора команд

Для настройки колонок списка команд необходимо вызвать команду контекстного меню Настройка списка (рис. 1.97).

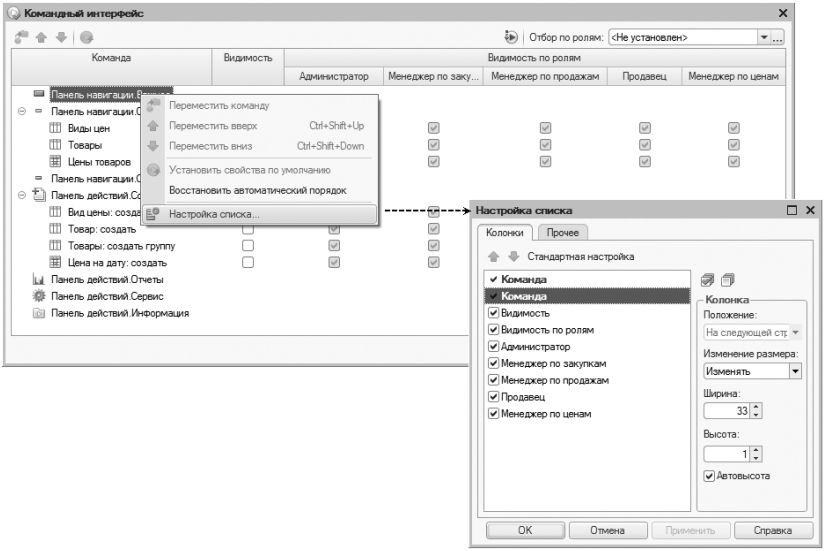


Рис. 1.97. Окно настройки отображения колонок редактора

В открывшемся окне можно настроить состав, размер и положение колонок списка команд.

Редактор командного интерфейса конфигурации

Редактор командного интерфейса конфигурации предназначен для редактирования порядка следования и видимости разделов в панели разделов.

Для открытия редактора можно из контекстного меню корня конфигурации выполнить команду Открыть командный интерфейс конфигурации. В результате откроется окно редактора, в котором отображается список разделов командного интерфейса, соответствующих подсистемам первого уровня иерархии с установленным свойством Включать в командный интерфейс (рис. 1.98).

Редактор позволяет:

- упорядочить разделы – в установленном порядке разделы будут показаны в панели разделов;
- установить общую видимость раздела и его видимость по ролям.

Приемы работы при настройке командного интерфейса конфигурации совпадают с приемами работы при настройке командного интерфейса подсистемы.

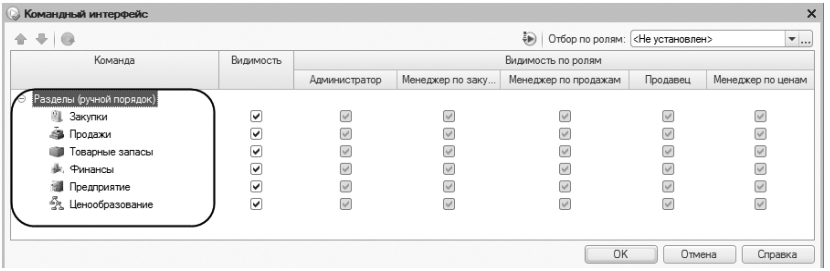


Рис. 1.98. Окно редактора командного интерфейса конфигурации

Редактор командного интерфейса основного раздела

Редактор командного интерфейса основного раздела предназначен для редактирования порядка следования и видимости команд основного раздела.

Для открытия редактора можно из контекстного меню корня конфигурации выполнить команду Открыть командный интерфейс основного раздела. В результате откроется окно редактора (рис. 1.99).

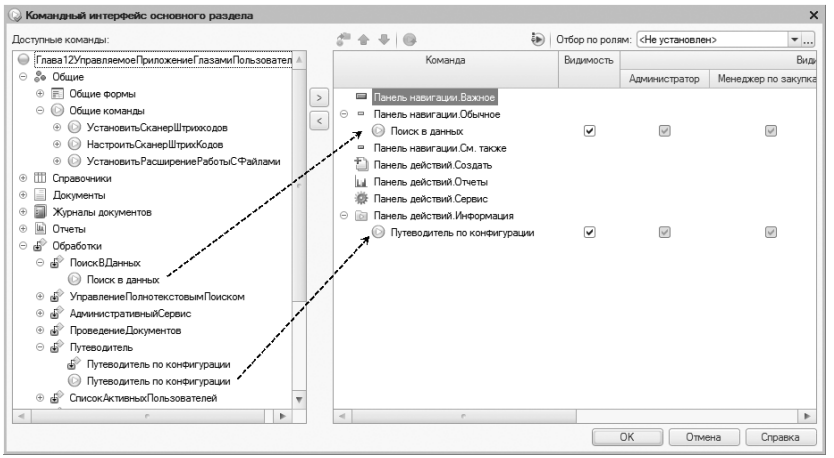


Рис. 1.99. Окно редактора командного интерфейса основного раздела

При редактировании командного интерфейса основного раздела в окне редактора отображается дополнительное табличное поле (слева). Это поле содержит иерархический список команд конфигурации, которые могут быть добавлены в командный интерфейс начальной страницы.

Если в конфигурации отсутствуют подсистемы, иерархический список команд не отображается.

В список доступных команд включаются:

- общие команды без параметров, для которых в свойстве Группа указаны панель и группа команд этой панели;
- стандартные команды объектов конфигурации, для которых установлено свойство Использовать стандартные команды;
- команды объектов конфигурации, определенные в подчиненной группе Команды.

Редактирование командного интерфейса выполняется в правом табличном поле редактора. Возможности и приемы редактирования командного интерфейса основного раздела полностью аналогичны возможностям и приемам редактирования командного интерфейса подсистемы.

Редактор «Все подсистемы»

Редактор «Все подсистемы» предназначен для управления подсистемами конфигурации, в том числе и их командными интерфейсами.

Для вызова редактора необходимо спозиционироваться на узле Подсистемы ветки Общие и из контекстного меню выбрать команду Все подсистемы (рис. 1.100).

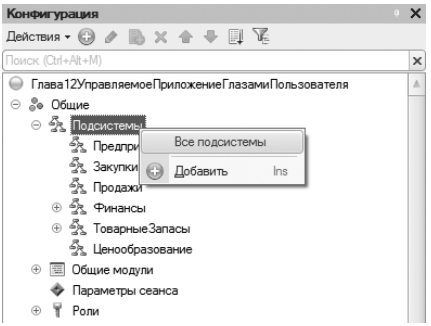


Рис. 1.100. Вызов редактора подсистем

В результате выполнения команды откроется окно редактора (рис. 1.101).

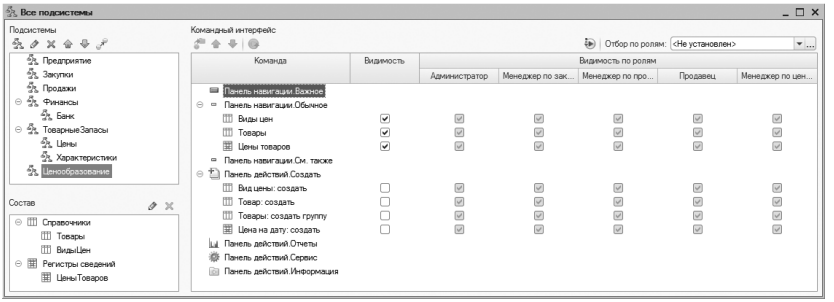


Рис. 1.101. Окно редактора подсистем

Редактор Все подсистемы визуально отличается от редактора командного интерфейса подсистемы наличием дополнительных табличных полей: Подсистемы и Состав (слева, сверху вниз).

В списке подсистем отображается дерево подсистем конфигурации. Используя командную панель этого списка (или контекстное меню), можно управлять подсистемами: добавлять, удалять, редактировать свойства, изменять порядок следования и иерархию (рис. 1.102).

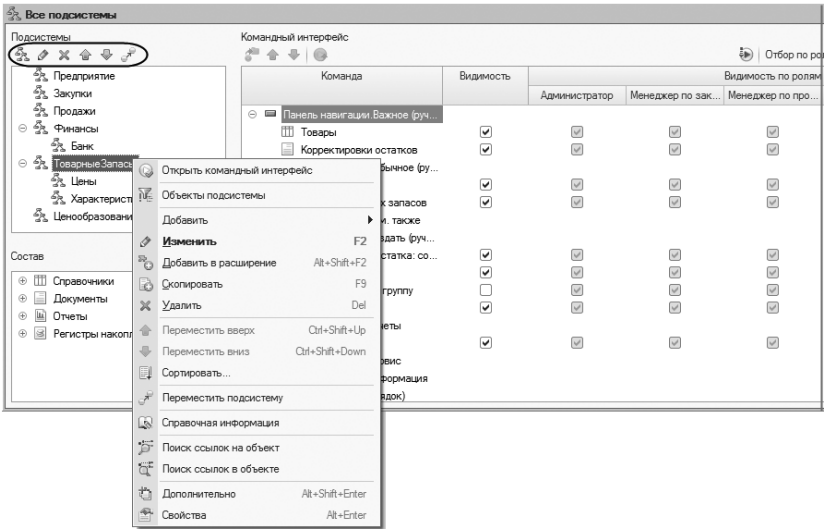


Рис. 1.102. Управление подсистемами

В списке Состав отображаются объекты конфигурации, включенные в подсистему, выбранную в дереве подсистем. Используя командную панель этого списка, можно управлять составом объектов, включенных в подсистему (рис. 1.103).

Командный интерфейс подсистемы, выбранной в дереве подсистем, редактируется в табличном поле Командный интерфейс (правое поле).

При необходимости, используя команду контекстного меню списка подсистем, из редактора подсистем можно открыть отдельный редактор командного интерфейса отдельной подсистемы. Приемы редактирования командного интерфейса одинаковы в обоих редакторах.

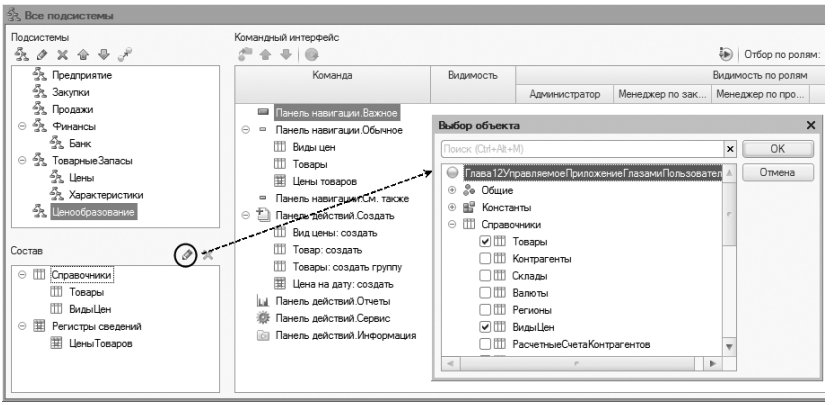


Рис. 1.103. Управление составом подсистемы

В редакторе Все подсистемы присутствует возможность перехода от команды к ее источнику в дереве объектов конфигурации. Для этого из контекстного меню списка команд нужно выбрать пункт Найти в дереве (рис. 1.104). В результате в дереве конфигурации будет выделен объект, предоставивший команду.

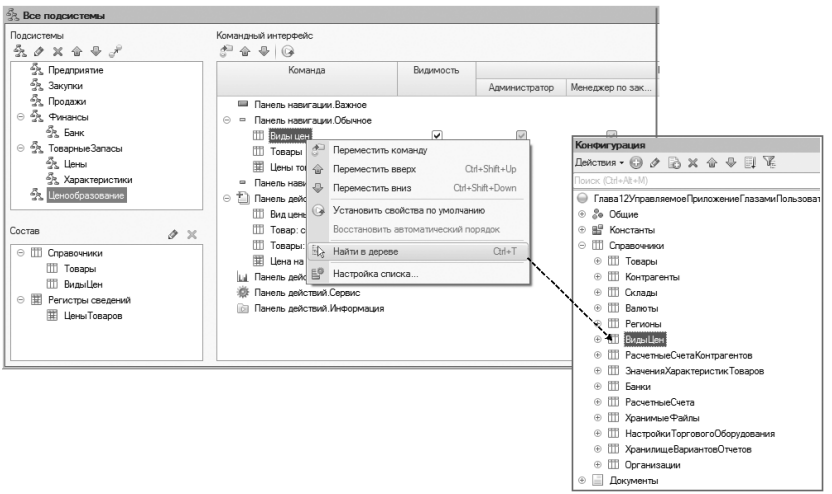


Рис. 1.104. Переход к источнику команды

Ручное размещение и видимость команд

А теперь вернемся к нашей задаче и настроим расположение и видимость команд в разделе Ценообразование.

Первое, что мы сделаем, – отобразим в командном интерфейсе команду назначения цены на товар. Назначение цены – это создание записи в регистре сведений Цены товаров.

Откроем редактор командного интерфейса подсистемы. В колонке Видимость для команды Цена на дату: создать установим флажок (рис. 1.105).

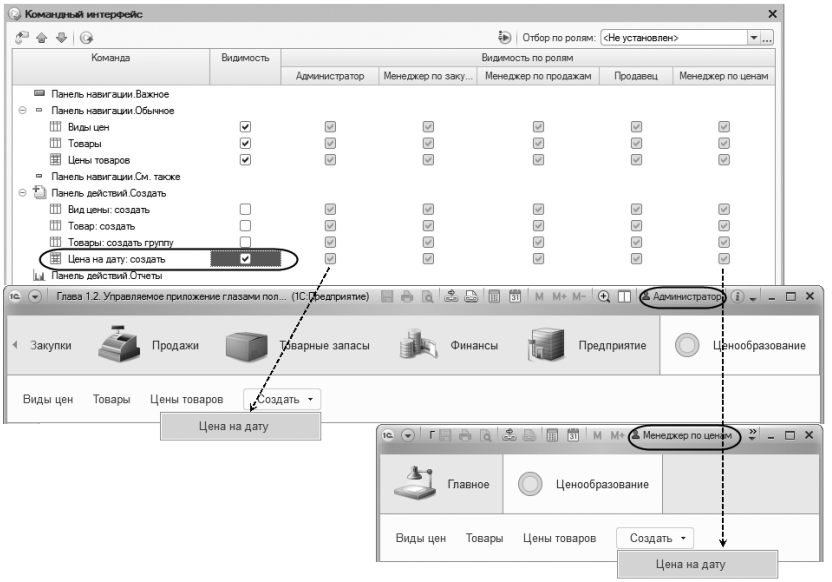


Рис. 1.105. Установка общей видимости по умолчанию для команды «Цена на дату: создать»

Теперь эта команда включена в командный интерфейс, причем для всех ролей (на рис. 1.105 показаны командные интерфейсы пользователей с ролями Администратор и Менеджер по ценам).

Далее необходимо скрыть команду открытия списка товаров для пользователей с ролью Менеджер по ценам. Для остальных ролей команда должна остаться доступной.

Для решения этой задачи воспользуемся настройкой видимости в разрезе ролей. Для команды Товары снимем флажок не в колонке Видимость, а в колонке Менеджер по ценам (рис. 1.106).

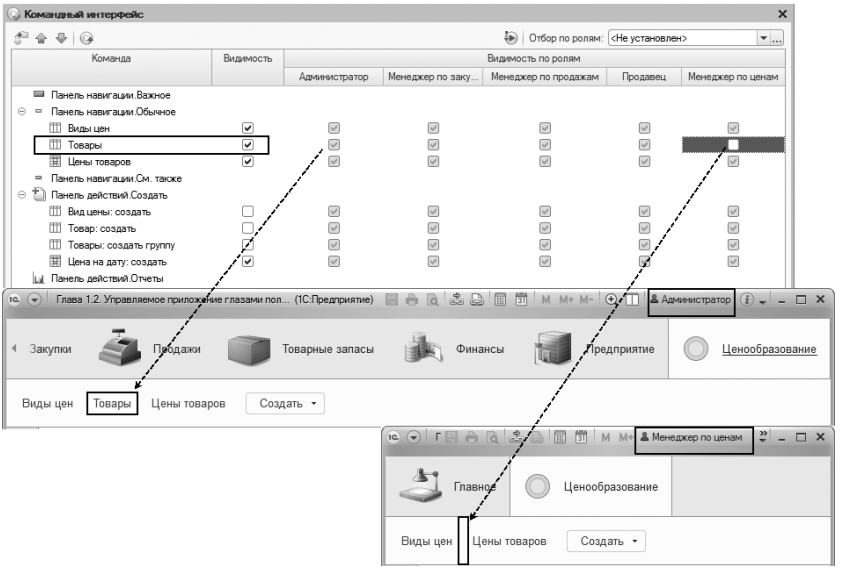


Рис. 1.106. Установка ролевой видимости по умолчанию для команды «Товары»

В результате для пользователя Администратор команда Товары включена в командный интерфейс, а для пользователя Менеджер по ценам команда не включена.

И последней настройкой мы изменим порядок следования команд в панели навигации. Сейчас все команды панели навигации расположены в группе ПанльНавигации.Обычное в алфавитном порядке. Нас такое расположение не устраивает. Команду Цены товаров переместим в группу ПанельНавигации.Важное (рис. 1.107).

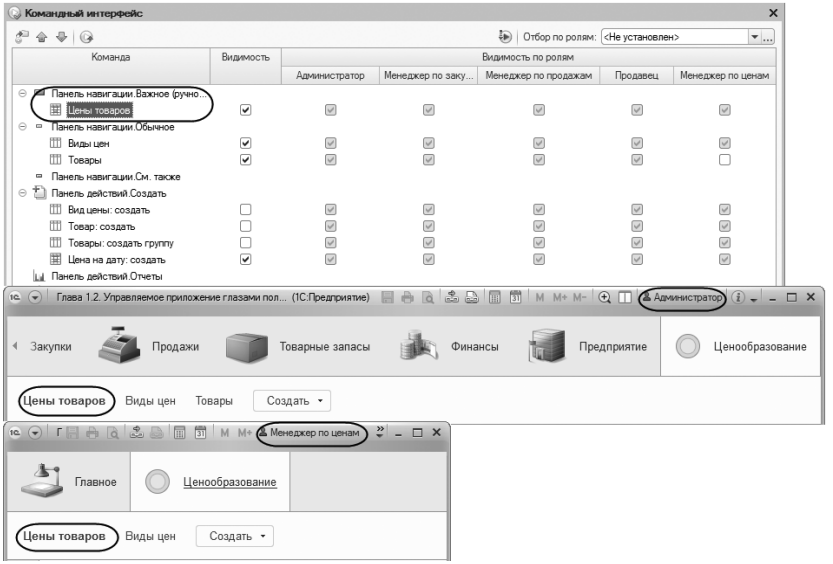


Рис. 1.107. Изменение группы для команды «Цены товаров»

В отличие от видимости порядок размещения команд в разрезе ролей не настраивается. И для пользователя Администратор, и для пользователя Менеджер по ценам команда Цены товаров располагается в одном и том же месте.

Глава 1.7. Влияние функциональных опций на командный интерфейс

Механизм функциональных опций позволяет на этапе эксплуатации прикладного решения подключать или отключать функциональные возможности конфигурации. Для того чтобы пользователь не видел команд, относящихся к неиспользуемым возможностям, система автоматически формирует состав доступных команд в зависимости от значений функциональных опций.

ВНИМАНИЕ!

В отличие от настройки доступности команд по ролям настройка по функциональным опциям является единой для всех ролей.

Механизм функциональных опций

Использование функциональных опций предполагает, что в прикладном решении присутствуют возможности, которые могут использоваться или не использоваться в зависимости от конкретных условий. Для каждой из таких возможностей разработчик создает в конфигурации функциональные опции и связывает с ними объекты, реализующие этот функционал.

Влияние же функциональных опций на командный интерфейс выражается в том, что система не включает в него команды тех объектов, которые относятся к выключенным опциям.

Для описания функциональных опций используются объекты конфигурации *Функциональные опции* и *Параметры функциональных опций*. Эти объекты располагаются в узлах ветки Общие дерева конфигурации (рис. 1.108).

Каждая функциональная опция в свойстве Состав хранит состав объектов конфигурации, которые зависят от функциональной опции, и в свойстве Хранение содержит объект, являющийся источником значения опции (рис. 1.109).

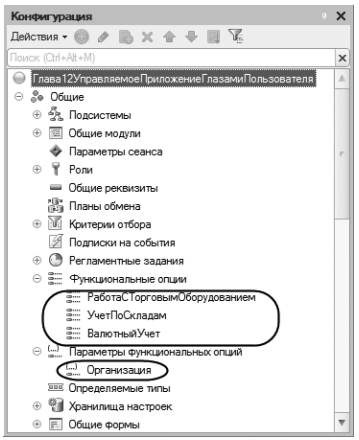


Рис. 1.108. Объекты конфигурации «Функциональные опции» и «Параметры функциональных опций»

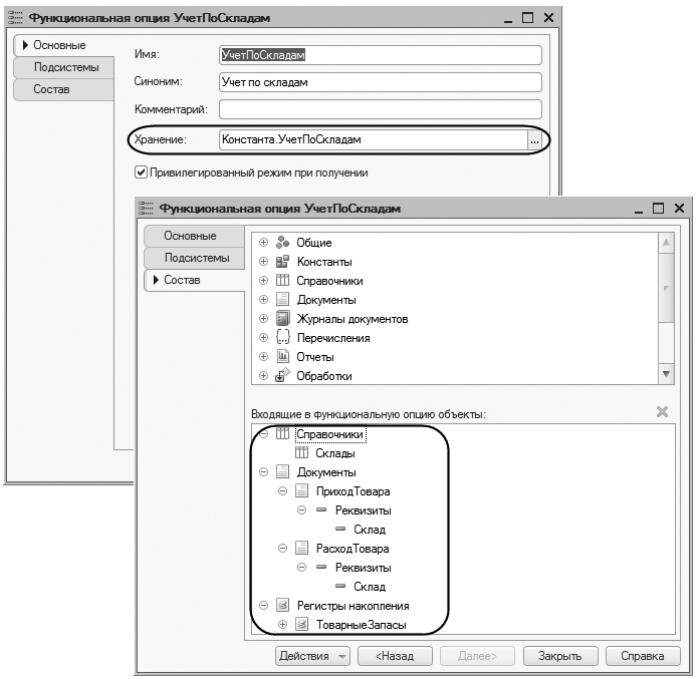


Рис. 1.109. Свойства объекта «Функциональная опция»

В зависимости от установленного для функциональной опции значения система включает в командный интерфейс (или исключает из него) команды объектов конфигурации, относящихся к этой опции.

Список объектов, которым можно назначить функциональные опции, приведен в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 5.5.10.4.

Состав объектов конфигурации, относящихся к функциональной опции, можно задать на закладке Состав окна редактирования объекта конфигурации или воспользоваться гиперссылкой Открыть свойства `Состав` палитры свойств (см. рис. 1.109).

В качестве источника значения функциональной опции могут использоваться:

- константа,
- реквизит справочника,
- ресурс регистра сведений.

Важно понимать, что значения функциональных опций не изменяют структуру конфигурации. Состав объектов конфигурации и соответствующих им информационных структур в базе данных остается неизменным при любых значениях функциональных опций.

В связи с тем, что функциональные опции отвечают на вопрос: «Использовать объект конфигурации или нет?» – их значения чаще всего имеют тип данных `Булево`.

Если значение функциональной опции хранится в константе, то затруднений в получении этого значения не возникает – система обращается к соответствующей константе.

Если же значение функциональной опции хранится в реквизите справочника или ресурсе регистра сведений, системе необходимо «рассказать», из какого элемента

В общем случае тип данных хранилища значения функциональной опции системой не ограничивается. Однако для автоматического управления командным интерфейсом используются только функциональные опции с булевым типом значений. Значения функциональных опций с другими типами доступны только для анализа из встроенного языка.

справочника или записи регистра сведений получать значения. То есть функциональную опцию необходимо *параметризовать*.

Фактически параметры функциональных опций задают систему координат, в которой хранятся значения функциональных опций.

Для параметризации функциональных опций используется объект конфигурации `Параметр функциональных опций`. Каждый из параметров определяет отдельную «координату» хранения значений функциональных опций. При этом один и тот же параметр может одновременно использоваться для параметризации нескольких функциональных опций.

Основываясь на значении параметра, система будет выбирать соответствующий элемент справочника или запись регистра сведений для получения значения функциональной опции.

Отключаем неиспользуемые команды

А теперь, познакомившись с функциональными опциями, вернемся к нашей задаче.

В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.6. Редактирование командного интерфейса». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.7. Влияние функциональных опций на командный интерфейс».

Нам требуется предоставить пользователям возможность самостоятельно настроить вариант задания цен на товары – либо в целом на товар, либо на товар в разрезе различных видов цен.

Для решения этой задачи в первую очередь создадим хранилище значения функциональной опции, значение которой будет управлять вариантом ценообразования. В качестве хранилища будем использовать константу `ЦенообразованиеПоВидамЦен` с типом данных `Булево`. Если значение константы `Истина`, то в подсистеме ценообразования используются виды цен, иначе виды цен не используются.

Используя контекстное меню узла Константы дерева конфигурации, добавим новую константу. В окне свойств установим значения для свойств добавленной константы (рис. 1.110):

- Имя – ЦенообразованиеПоВидамЦен,
- Синоним – оставляем автоматически сформированный «Ценообразование по видам цен»,
- Тип – константа имеет тип данных Булево,
- Использовать стандартные команды – нет.

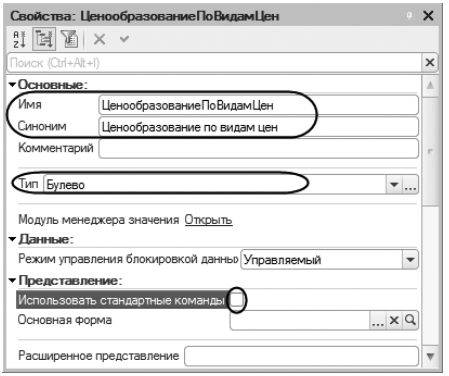


Рис. 1.110. Заполнение свойств константы

Обеспечим возможность редактирования значения константы. Для этого найдем в ветке Общие формы форму констант Общие настройки и откроем ее.

В дерево элементов формы добавим поле Ценообразование по видам цен, связанное с реквизитом формы ЦенообразованиеПоВидамЦен. Для этого перетащим мышью этот реквизит из окна реквизитов (в правом верхнем списке) в окно элементов формы в левом верхнем списке (рис. 1.111).

Мы обеспечили возможность хранения и редактирования значения функциональной опции. А теперь давайте представим, что мы изменили значение константы.

Более подробно о работе с редактором форм можно прочитать в главе 2.3 на стр. 256.

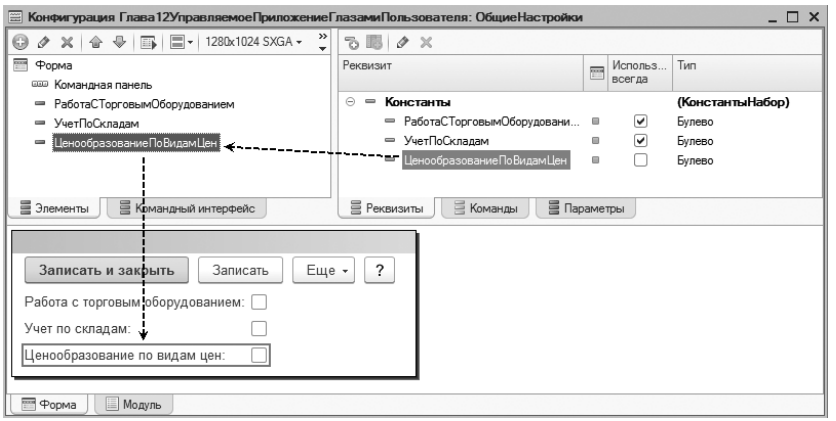


Рис. 1.111. Добавление новой константы в форму

Что хотелось бы увидеть в командном интерфейсе? Правильно, хочется увидеть обновленный командный интерфейс (панель разделов, панель навигации, начальную страницу и т. д.), построенный с учетом нового значения функциональной опции. Для этого в модуле формы реализован обработчик события `ПослеЗаписи()`, в котором методом глобального контекста `ОбновитьИнтерфейс()` выполняется обновление интерфейса пользователя (рис. 1.112).

```
// Обновляет интерфейс приложения после записи общих настроек
&НаКлиенте
Процедура ПослеЗаписи()
    ОбновитьИнтерфейс();
КонецПроцедуры
```

Рис. 1.112. Обработчик «ПослеЗаписи()» в модуле формы

Теперь добавим саму функциональную опцию. Для этого используем контекстное меню узла Функциональные опции дерева конфигурации. В окне редактирования свойств установим значения для свойств добавленной функциональной опции (рис. 1.113):

- Имя – ЦенообразованиеПоВидамЦен;
- Синоним – оставляем автоматически сформированный «Ценообразование по видам цен»;

- Хранение – указываем константу ЦенообразованиеПоВидамЦен, хранящую значение функциональной опции.

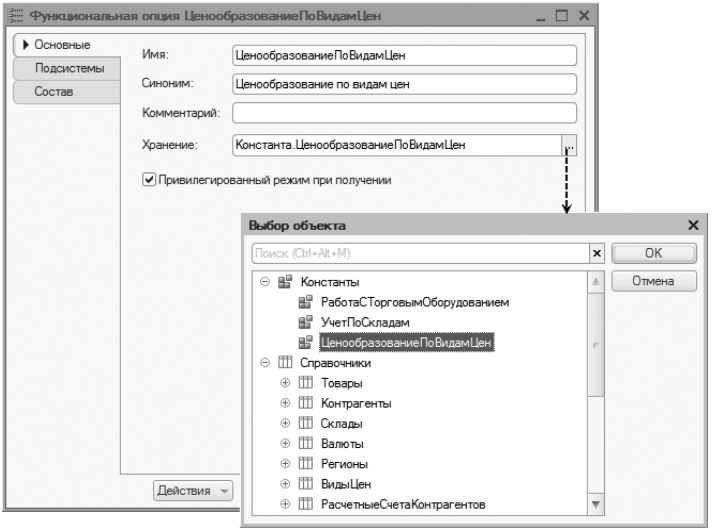


Рис. 1.113. Заполнение свойств функциональной опции

Следующим шагом будет выбор объектов конфигурации, зависящих от созданной функциональной опции.

Для этого в окне редактирования свойств функциональной опции перейдем на закладку Состав и в дереве объектов конфигурации пометим справочник ВидыЦен и измерение ВидЦен регистра сведений ЦеныТоваров (рис. 1.114).

Если ценообразование в разрезе видов цен не используется, то все команды для работы со справочником ВидыЦен будут «лишними». Поэтому зависимость настраиваем для справочника в целом.

С регистром сведений ситуация другая. Нам в любом случае необходимо выполнять ценообразование в разрезе товаров. А вот в разрезе видов цен ценообразование выполняется не всегда. Поэтому зависимость настраиваем не в целом для регистра сведений, а для подчиненного ему объекта измерения.

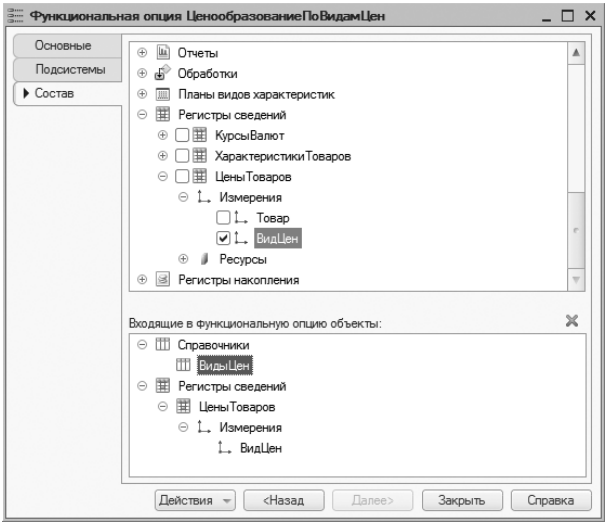


Рис. 1.114. Заполнение состава функциональной опции

В общем случае одни и те же объекты конфигурации могут входить в состав нескольких функциональных опций. В такой ситуации доступность их команд определятся по правилу логического ИЛИ – если хотя бы одна функциональная опция имеет логическое значение Истина, команды будут доступны.

Если же объект конфигурации не входит в состав ни одной из функциональных опций, то и доступность его команд от опций не зависит.

Обновим конфигурацию базы данных и запустим демонстрационную базу в режиме 1С:Предприятие от имени пользователя Администратор.

В панели разделов основного окна выберем раздел Ценообразование. Команды обращения к списку справочника Виды цен в командном интерфейсе нет. Эта команда отключена вследствие того, что константа Ценообразование по видам цен имеет значение Ложь (рис. 1.115).

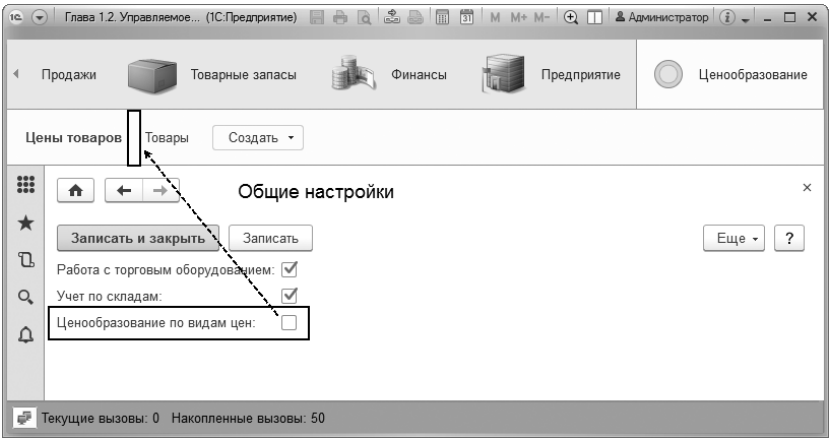


Рис. 1.115. При отключенной функциональной опции команда «Виды цен» в командном интерфейсе отсутствует

Выберем команду Цены товаров, открывающую список регистра сведений Цены товаров (рис. 1.116).

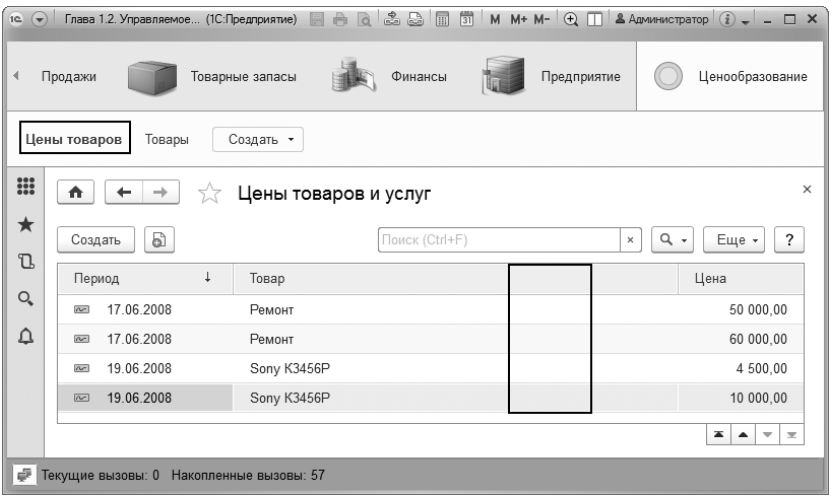


Рис. 1.116. Список регистра сведений «Цены товаров» при отключенной функциональной опции

В списке отсутствует измерение Вид цен регистра сведений Цены товаров.

Теперь в форме Общие настройки (доступной из меню функций раздела Предприятие) у константы Ценообразование по видам цен установим признак использования и нажмем кнопку Записать. Посмотрим теперь, как изменился состав команд панели навигации и вид формы списка регистра сведений (рис. 1.117).

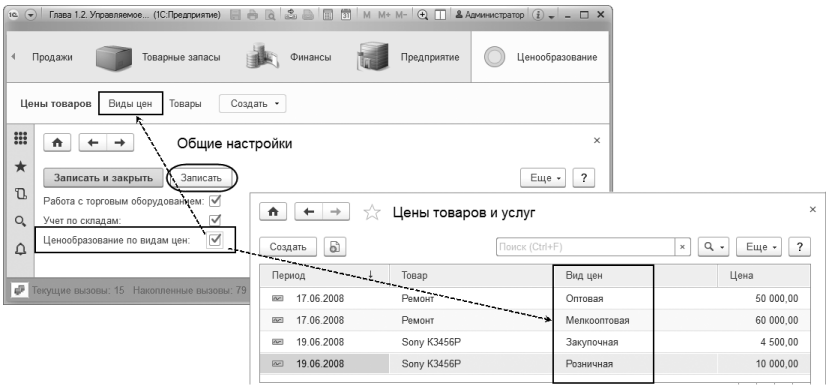


Рис. 1.117. Командный интерфейс и список регистра сведений «Цены товаров» при включенной функциональной опции

В панели навигации появилась команда Виды цен, открывающая форму списка справочника, а в форме списка регистра сведений отображается колонка Вид цен.

Таким образом, при отсутствии учета цен в разрезе их видов мы имеем возможность убрать из командного интерфейса «лишние» команды.

Глава 1.8. Пользовательская настройка интерфейса

Мы уже говорили о том, что пользователь имеет возможность настроить глобальный командный интерфейс в соответствии с собственными предпочтениями. Каждый пользователь прикладного решения может настроить состав команд в области системных команд и команды в панелях основного окна приложения.

Настройки, выполненные пользователем, сохраняются между сеансами его работы с прикладным решением. Пользовательские настройки перекрывают настройки, сделанные разработчиком в конфигураторе, которые пользователь видит по умолчанию при старте прикладного решения, – за исключением случаев, когда разработчик программно изменяет сохраненные пользовательские настройки (этот вариант в книге мы рассматривать не будем).

Настройка области системных команд

Состав команд, отображаемый в области системных команд основного окна приложения, настраивается вызовом подменю Добавить или удалить кнопки системного меню Другие кнопки (рис. 1.118).

Пользователь может отметить в списке те команды, которые он хочет видеть в области системных команд. Для возврата к настройкам по умолчанию необходимо выбрать в подменю команду Сброс панели (см. рис. 1.118).

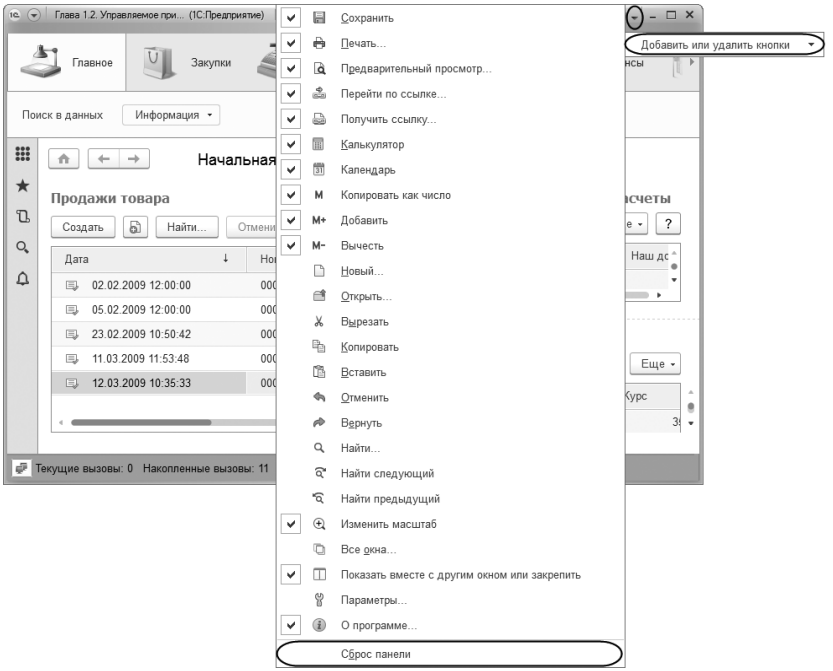


Рис. 1.118. Меню настройки состава командных панелей главного меню

Настройка командного интерфейса

В основном окне приложения пользователь может настраивать состав отображаемых команд и их расположение в панели разделов, панели функций и в меню функций текущего раздела.

Пользовательская настройка командного интерфейса выполняется в диалогах, вызываемых из подменю Вид главного меню системы (рис. 1.119).

Диалог настройки панели разделов (о котором будет рассказано ниже) вызывается по команде главного меню Вид – Настройка панели разделов. Кроме того, из этого подменю пользователь может вызвать диалог настройки панелей интерфейса (подробнее см. раздел «Настройка панелей интерфейса» на стр. 23) и диалог настройки начальной страницы (подробнее см. главу 2.10 «Начальная страница» на стр. 422).

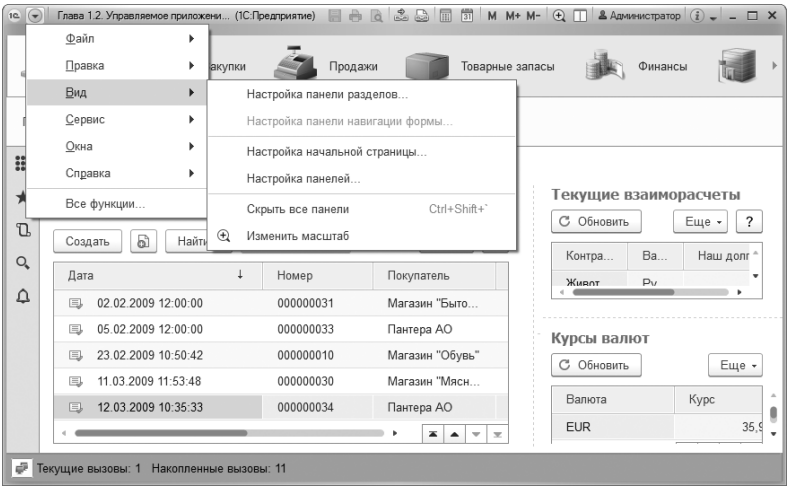


Рис. 1.119. Команды открытия окон настройки интерфейса

Если пользователь открыл какое-нибудь окно клиентского приложения, имеющее панель навигации, то он может настроить состав команд в панели навигации этого окна командой Вид – Настройка панели навигации формы.

Настроить команды раздела пользователь может с помощью команд Настройка навигации и Настройка действий из меню функций раздела (рис. 1.120).

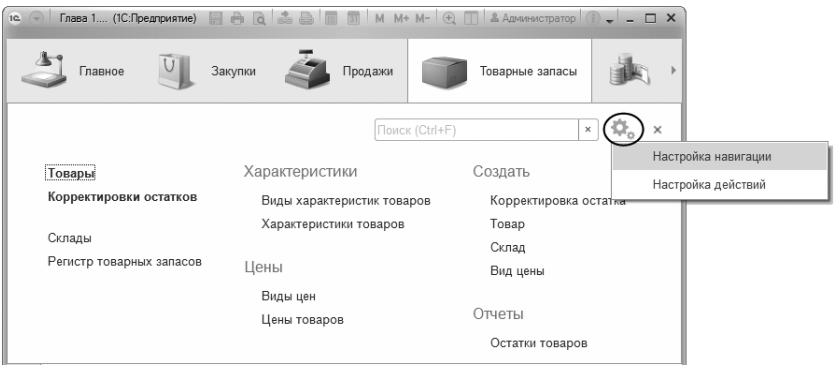


Рис. 1.120. Вызов меню настроек из меню функций раздела

Диалог настройки любой командной панели содержит два списка: доступные команды и выбранные команды (рис. 1.121).

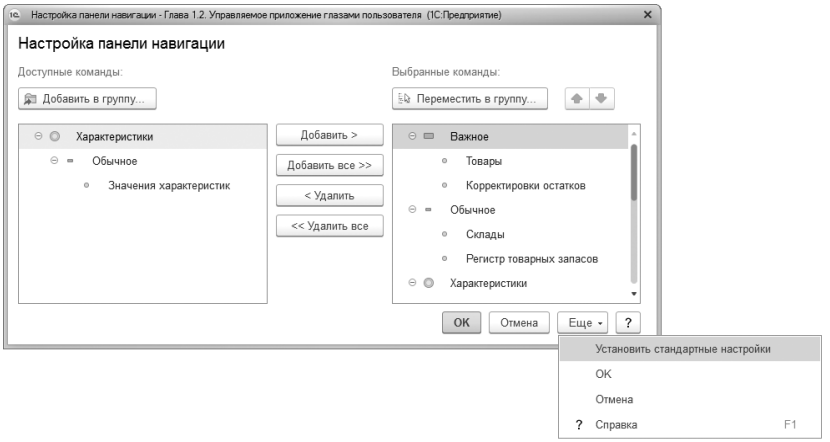


Рис. 1.121. Диалог настройки командной панели

Доступность команд определяется ролью пользователя и функциональными опциями. В диалоге отображаются только те команды, которые связаны с доступными пользователю элементами командного интерфейса, и эти элементы не относятся к отключенным функциональным опциям.

В командном интерфейсе отображаются только те команды, которые пользователь поместил в список выбранных. Команды отображаются в тех группах и в том порядке, которые установлены в этом списке.

Перемещение выбранной команды между списками выполняется кнопками Добавить >, Добавить все >>, Удалить > и Удалить все >>.

Для помещения команды в группу, отличающуюся от стандартной, используются кнопки Добавить в группу... и Переместить в группу...

Порядок команд в группах устанавливается кнопками перемещения вверх и вниз (кнопки со стрелками в командной панели списка Выбранные команды).

В диалоге настройки поддерживаются множественное выделение команд и перетаскивание команд с помощью мыши.

Применение выполненной настройки происходит с помощью кнопки OK. При этом сделанные настройки сохраняются в информационной базе, а окно настроек закрывается. Для отмены выполненной настройки необходимо выбрать кнопку Отмена.

При последующем отображении командного интерфейса к нему применяются последние сохраненные настройки.

Для возврата к стандартным (заданным в конфигурации) настройкам необходимо в подменю Еще выбрать команду Установить стандартные настройки (см. рис. 1.121). Для применения стандартных настроек их также необходимо подтвердить кнопкой OK.

Настройка панели разделов

В диалоге настройки панели разделов отображаются только доступные пользователю команды выбора раздела. Для панели разделов можно настроить (рис. 1.122):

- видимость и порядок разделов,
- способ отображения разделов.

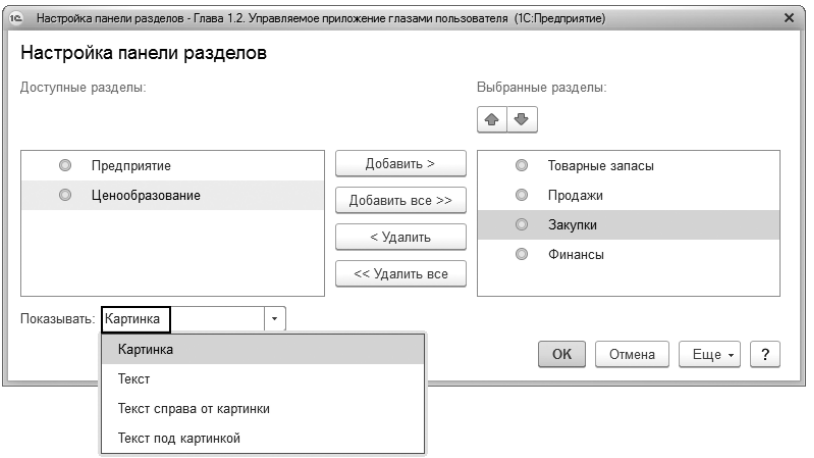


Рис. 1.122. Настройка панели разделов

Видимость и порядок разделов настраиваются аналогично общему порядку, описанному ранее.

Для настройки способа отображения раздела необходимо в поле выбора Показывать указать один из режимов:

- Картинка – каждый раздел показывается только картинкой;
- Текст – каждый раздел показывается только текстом (название раздела);
- Текст справа от картинки – каждый раздел показывается картинкой и текстом, при этом название раздела выводится справа от картинки;
- Текст под картинкой – каждый раздел показывается картинкой и текстом, при этом название раздела выводится под картинкой.

После закрытия диалога настройки кнопкой OK панель разделов будет изменена в соответствии с настройками, показанными на предыдущем рисунке (рис. 1.123).

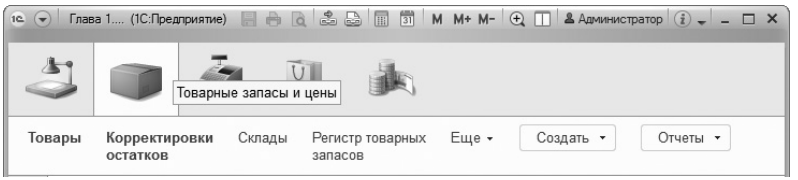


Рис. 1.123. Настроенная панель разделов

Если пользователем скрыты все разделы (в диалоге настройки список Выбранные разделы пустой), то панель разделов также скрывается. Если при этом в разделе Основной раздел есть команды, то они отображаются в панели функций (рис. 1.124).

Рис. 1.124. Основное окно с неотображаемой панелью разделов

Настройка меню функций текущего раздела

Из меню функций раздела с помощью команды Настройка навигации (см. рис. 1.120) пользователь может настроить состав и порядок команд, идущих вне групп в начале панели функций или в меню функций раздела (рис. 1.125).

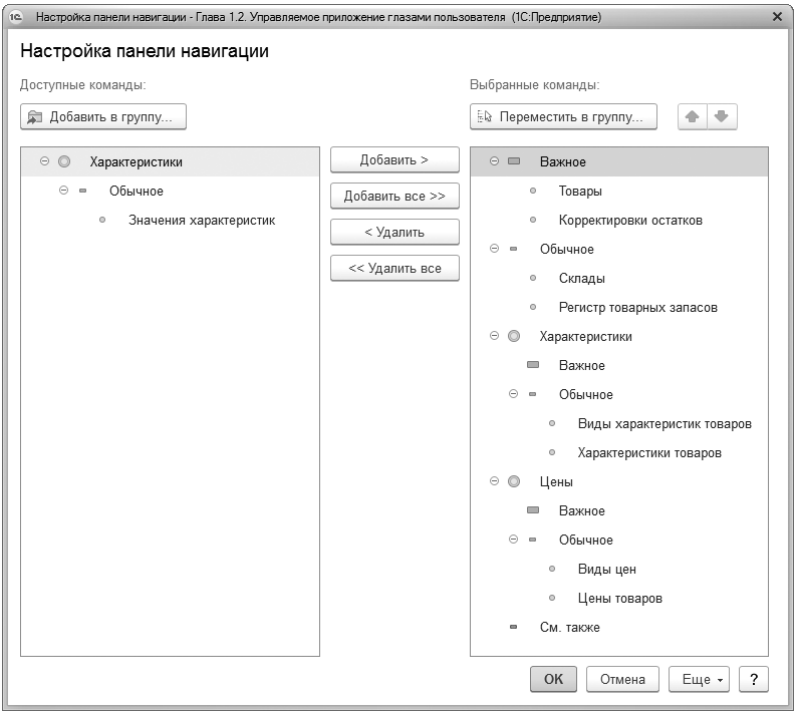


Рис. 1.125. Настройка панели функций раздела

В списке Выбранные команды команды отображаются с учетом иерархии подсистем выбранного раздела и иерархии групп.

Видимость и порядок команд настраиваются аналогично общему порядку, описанному ранее.

Перемещение команд между подчиненными подсистемами (и подсистемой верхнего уровня иерархии, соответствующей разделу) невозможно.

Из меню функций раздела с помощью команды Настройка действий (см. рис. 1.120) можно настроить команды, собранные в стандартные группы в панели функций или в меню функций раздела. Пользователь может настроить видимость и расположение команд (рис. 1.126):

- открытия форм новых объектов в стандартной группе Создать,
- открытия отчетов в стандартной группе Отчеты,
- вызова обработок в стандартной группе Сервис.

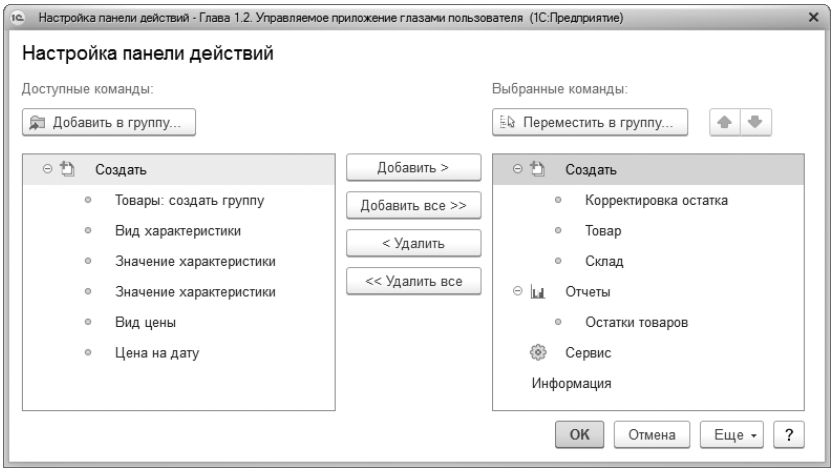


Рис. 1.126. Настройка панели функций раздела

Видимость и порядок команд настраиваются аналогично общему порядку, описанному ранее.

В списке Выбранные команды вложенные подсистемы не отображаются, так как все группы являются общими. Команды раздела и всех его подразделов отображаются в одном списке.

Одновременное отображение двух окон

Для решения различных экономических задач в «1С:Предприятии» часто требуется возможность одновременного просмотра на экране сразу двух окон. Например, для сравнения отчетов за разные периоды. Или для сравнения двух документов.

Это можно сделать двумя способами:

- с помощью закрепления одного из окон,
- с помощью объединения двух окон.

При закреплении одного из окон рабочая область основного окна приложения сдвигается в сторону, чтобы освободить место для закрепленного окна. Новые окна будут по-прежнему открываться в рабочей области, рядом с закрепленным окном, не закрывая его. Таким способом удобно, например, сравнивать данные отчета и документов, на основании которых он был сформирован. Или просто открывать разные документы из закрепленного рядом списка.

При объединении двух окон эти окна становятся как бы одним целым и показываются вместе, когда любое из них активизируется. А новые окна, как и раньше, открываются, занимая всю рабочую область, поверх этих объединенных окон. Таким способом удобно, например, сравнивать результаты одного и того же отчета за разные периоды или просто два открытых документа.

Чтобы закрепить окно (слева/справа/сверху/снизу), можно воспользоваться командой Показать вместе с другим окном или закрепить из области системных команд (рис. 1.127).

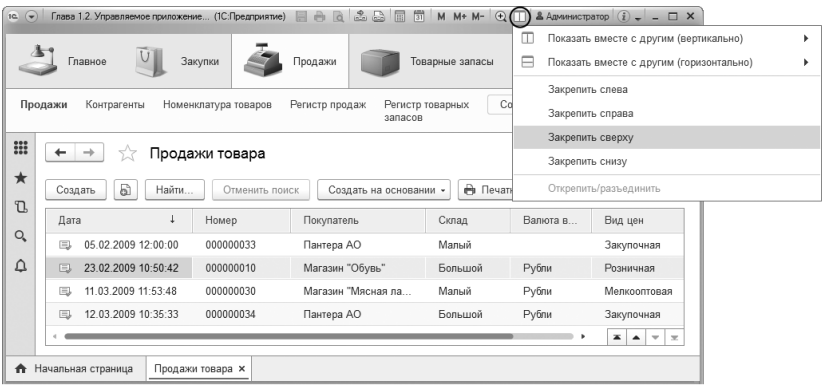


Рис. 1.127. Закрепление окна из области системных команд

Например, пользователь может закрепить список продаж товаров в верхней части основного окна приложения, а на оставшемся

месте открывать интересующие его документы продаж. При этом документы будут располагаться под закрепленным списком, не перекрывая его (рис. 1.128).

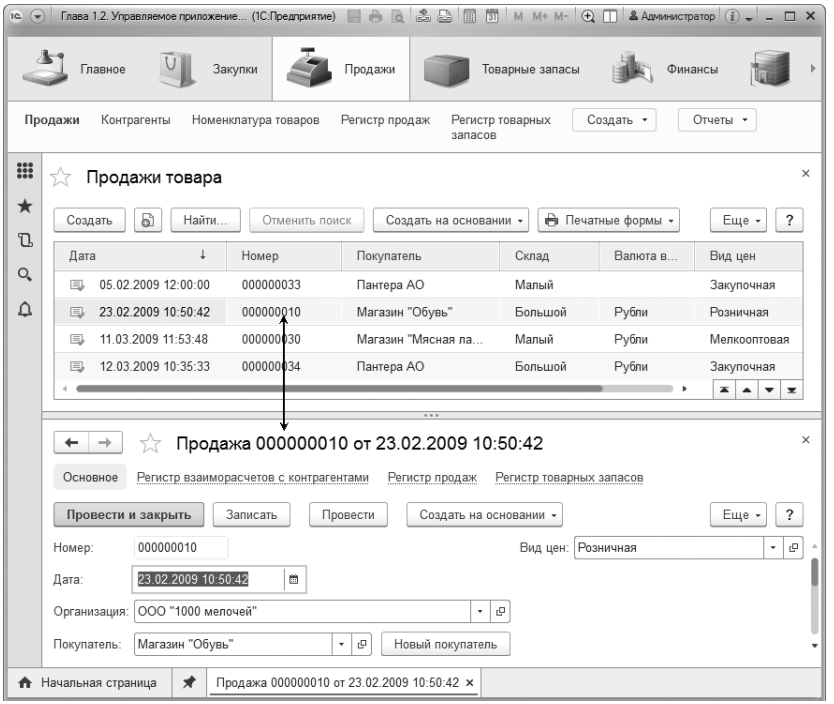


Рис. 1.128. Открытие документов из закрепленного списка документов

Или же, например, пользователь хочет сравнить данные отчетов, сформированных за разные периоды. В этом случае удобно сформировать оба отчета в разных окнах и объединить их из панели открытых.

Чтобы открыть тот же отчет в другом окне, нужно выполнить команду Открыть в новом окне из подменю Еще формы отчета. Затем сформировать его за другой период и вызвать команду Показать вместе с текущим (вертикально/горизонтально) из контекстного меню заголовка неактивного первого окна (отчета, с которым надо сравнивать) в панели открытых (рис. 1.129).

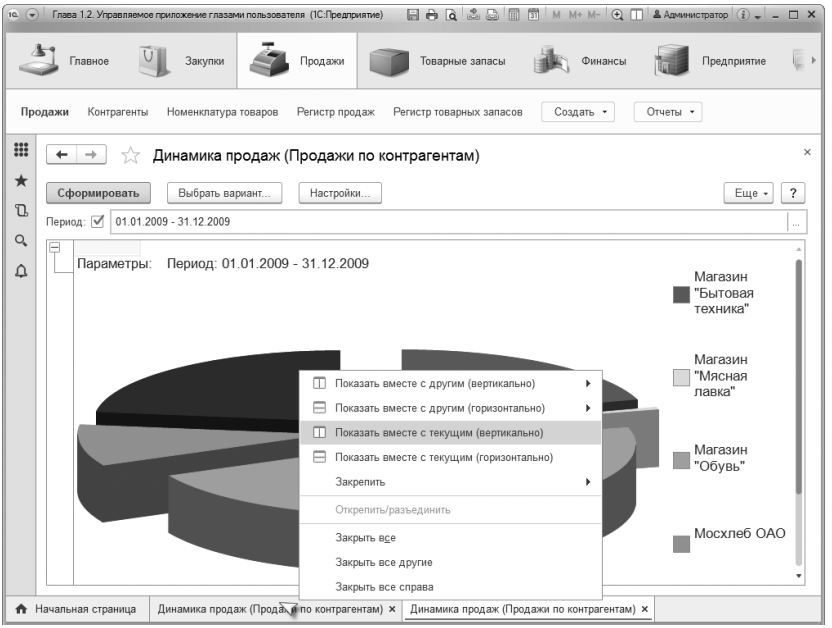


Рис. 1.129. Объединение окон из панели открытых

В результате оба окна будут объединены и показаны рядом друг с другом (рис. 1.130). Если поверх них будут открыты еще какие-то окна, то при активизации любого из объединенных окон оба отчета будут показываться вместе.

Объединить/закрепить окна можно также с помощью команд контекстного меню заголовка окна клиентского приложения. При объединении окон нужно выбрать любое из открытых окон (кроме начальной страницы) для объединения (рис. 1.131).

Кроме того, закрепление и объединение окон можно выполнять с помощью команд главного меню.

При объединении или закреплении действует общее правило: если форма, над которой выполняется команда объединения или закрепления, уже является объединенной или закрепленной, то вначале выполняется команда разъединения или открепления, а затем – выбранная команда.

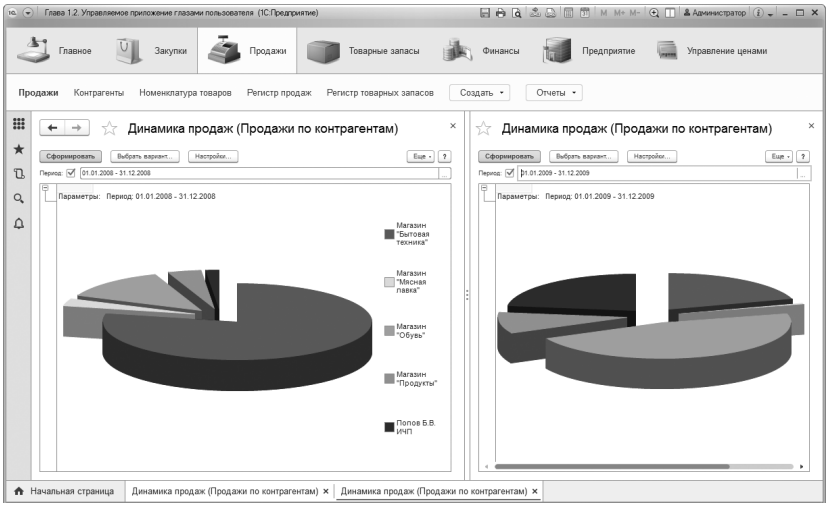


Рис. 1.130. Сравнение результатов отчетов за разные периоды

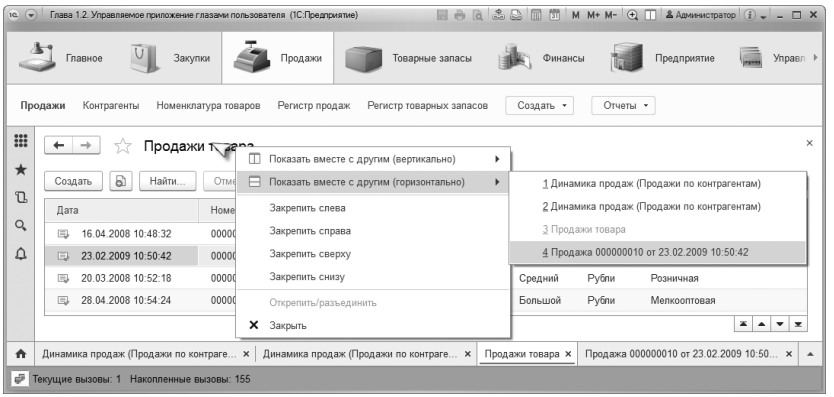


Рис. 1.131. Объединение/закрепление окон из контекстного меню заголовка окна

Кроме того, открепить окно или разъединить окна в любой момент можно самостоятельно с помощью команды Открепить/разъединить, которая присутствует во всех показанных меню.

Настройка масштаба форм приложения

В «1С:Предприятии» существует возможность быстрого масштабирования форм, с помощью которой каждый пользователь в зависимости от уровня своего зрения или других соображений может легко и быстро увеличить/уменьшить масштаб отображения одной или сразу всех форм приложения.

Для этого можно воспользоваться командой Изменить масштаб из области системных команд или аналогичной командой главного меню из подменю Вид. Диалог для изменения масштаба можно мышью перетащить в любое удобное место основного окна приложения (рис. 1.132).

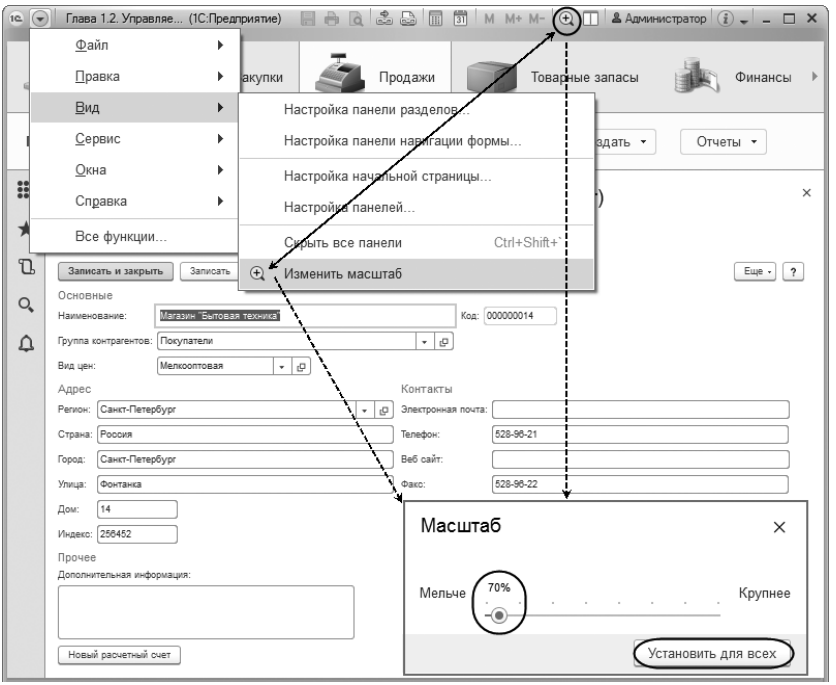


Рис. 1.132. Настройка масштаба форм

Перетащив ползунок, можно изменить масштаб текущей открытой формы и сразу же оценить результат. При нажатии кнопки Установить для всех такой же масштаб будет установлен для всех форм приложения.

Масштабы хранятся для каждой формы отдельно в локальном хранилище на компьютере пользователя. При установке нового масштаба для всех форм отдельные значения для каждой формы удаляются.

Кроме того, масштаб формы можно задать в конфигураторе с помощью свойства Масштаб (подробнее об этом будет рассказано во второй части в разделе «"Вариант масштаба", "Масштаб"» на стр. 275), а также масштаб формы можно изменить из встроенного языка.

Глава 1.9. Настраиваем представление команд

Кроме настройки размещения и видимости команд необходимо обратить внимание и на пользовательское представление команд. Они должны быть представлены в командном интерфейсе «говорящими» названиями: представление команды должно помочь пользователю быстро вспомнить ее назначение.

Правила автоматического формирования представления приведены в документации «1С:Предприятие 8.3.10. Руководство разработчика», (приложение 3 «Правила формирования текстов стандартных команд и автоматических заголовков форм»).

Если разработчик специально не настраивает представление, то при формировании командного интерфейса система автоматически формирует представление для команд.

Для задания различных представлений у многих объектов реализованы специальные свойства. Состав этих свойств определяется прототипом объекта конфигурации. Свойства отображаются на закладке Основные окна редактирования свойств объекта конфигурации (рис. 1.133) или в палитре свойств в группе Представление.

Задавая значения свойствам из этой группы, разработчик может формировать представления для заголовков форм объекта, представления для стандартных команд объекта и представления для подсказок к командам.

ВНИМАНИЕ!

Заполнение свойств, связанных с представлением, необходимо только в тех случаях, когда синоним не может представить объект требуемым образом или есть необходимость в уточнении информации, отображаемой по умолчанию.

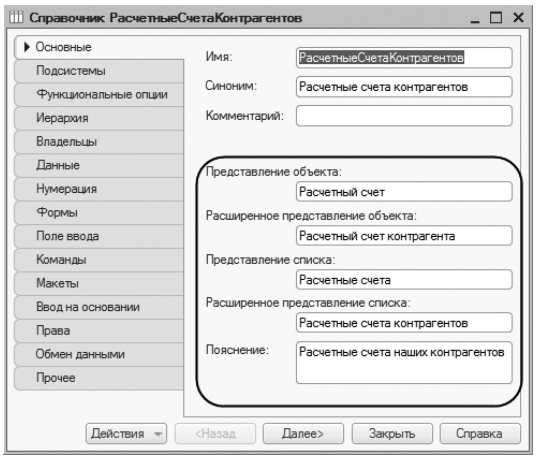


Рис. 1.133. Интерфейсные свойства объектов конфигурации

Если не заданы все свойства объекта конфигурации, используемые в формировании представления, то используется его системное текстовое представление – в большинстве случаев это значение его свойства Синоним.

Это свойство должно быть заполнено (по умолчанию свойство заполняется исходя из имени объекта), и значение свойства должно осмысленно и максимально лаконично описывать объект конфигурации.

> Если синоним не задан, то для представления объекта конфигурации используется значение его свойства Имя.

В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.7. Влияние функциональных опций на командный интерфейс». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.9. Настраиваем представление команд».

Например, в демонстрационной базе для справочника Товары заполнены свойства:

- Представление объекта – значение «Товар», представляет объект данных;

- Представление списка – значение «Товары», представляет множество (список) объектов данных.

Команды открытия списка, создания объекта и создания группы представлены в интерфейсе в соответствии с заданным представлением (рис. 1.134).

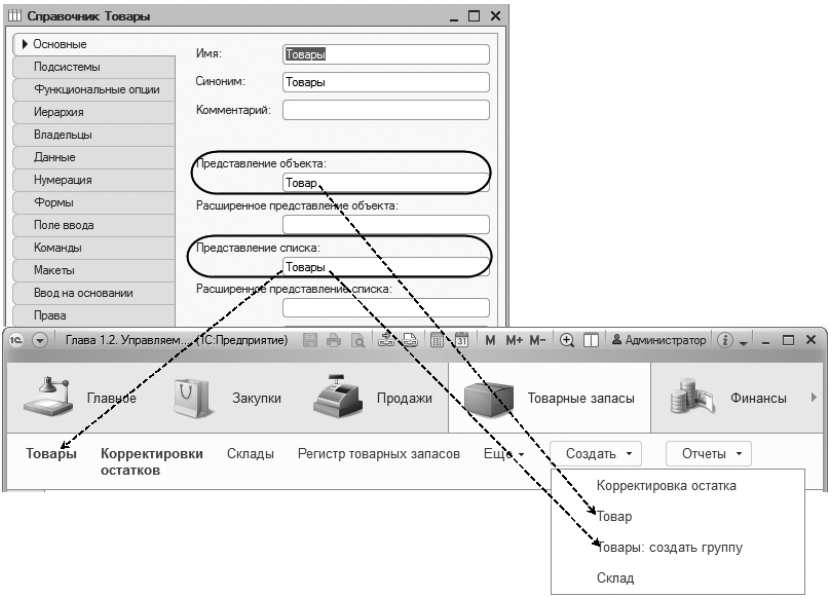


Рис. 1.134. Представление команд справочника «Товары», первый вариант

Нас будут интересовать свойства, влияющие на представление стандартных команд объектов. К этим свойствам относятся:

- Представление объекта – определяет представление стандартной команды создания объекта;
- Представление списка – определяет представление стандартной команды открытия списка объектов;
- Пояснение – определяет подсказку к стандартным командам.

Для всех стандартных команд объекта используется один и тот же текст пояснения.

Для демонстрации влияния свойств, связанных с представлением стандартных команд, на командный интерфейс изменим свойства следующим образом:

- Представление объекта – зададим значение «Номенклатурная позиция»;
- Представление списка – зададим значение «Номенклатура товаров»;
- Пояснение – зададим значение «Номенклатура товаров, продаваемых компанией».

Свойства изменены в демонстрационных целях. В подавляющем большинстве случаев представление Товар для объекта и представление Товары для списка являются вполне приемлемыми.

Запустив конфигурацию в режиме 1С:Предприятие, мы видим, что представление команд в интерфейсе изменилось в соответствии с новыми значениями (рис. 1.135).

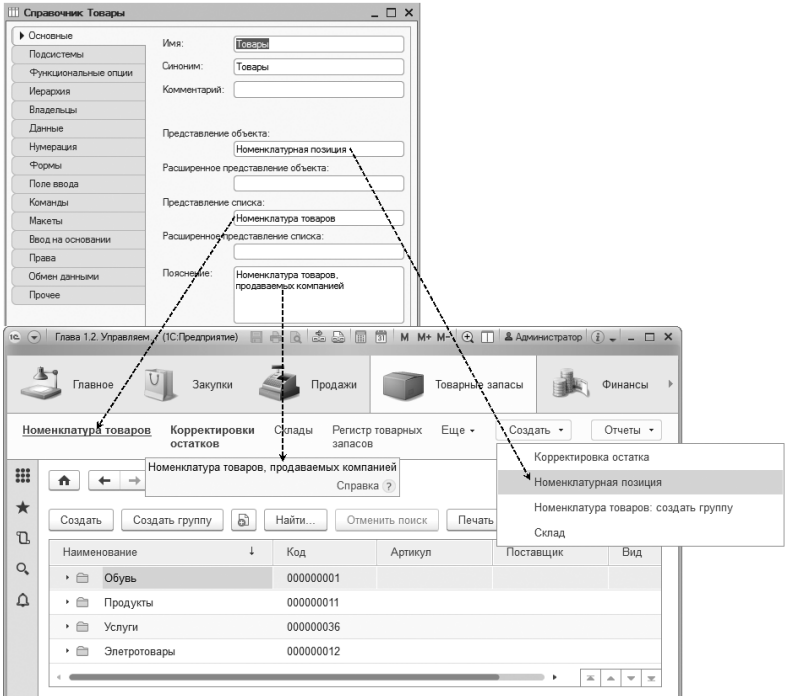


Рис. 1.135. Представление команд справочника «Товары», второй вариант

Текст, заданный в «представительских» свойствах, представляет не только команды, но и формы. Например, для справочника в заголовке формы списка используется значение свойства Представление списка (рис. 1.136 вверху), а в заголовке формы элемента, помимо текстового представления самого объекта, еще и значение свойства Представление объекта (рис. 1.136 внизу).

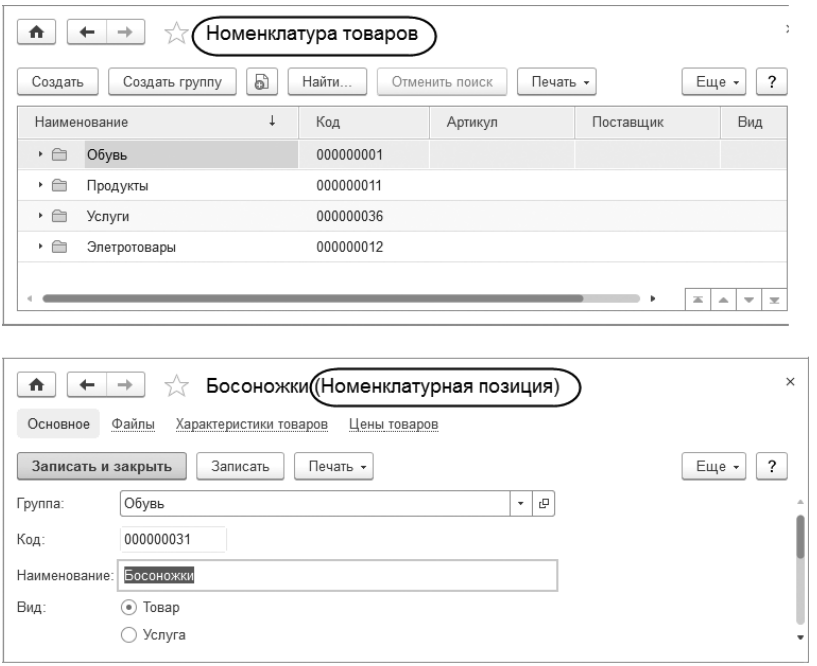


Рис. 1.136. Представление заголовка формы списка и заголовка формы элемента

Таким образом, при заполнении свойств, связанных с формированием представления объектов и команд в интерфейсе, необходимо выбирать для них такие значения, которые и были бы информативны, и одинаково читались как в представлении команд, так и в заголовках форм.

Для подсистем также определены свойства, позволяющие настроить их представление в интерфейсе. Настроим представление созданной нами подсистемы Ценообразование.

На данный момент раздел, соответствующий подсистеме, представлен картинкой по умолчанию (желтый диск) и значением свойства Синоним подсистемы (рис. 1.137).

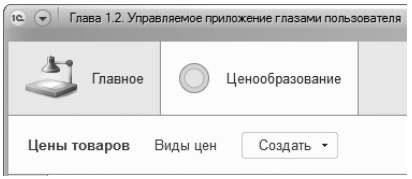


Рис. 1.137. Представление раздела по умолчанию

Хотелось бы назначить разделу собственную картинку и вместо текста Ценообразование выводить текст «Управление ценами».

Для настройки представления обратимся к свойствам подсистемы Ценообразование и зададим значения для свойств, определяющих представление подсистемы (рис. 1.138):

- для свойства Синоним зададим значение «Управление ценами»,
- в свойстве Пояснение введем текст «Выполнение задач по назначению и редактированию цен на товары»,
- в свойстве Картинка выберем картинку ПодсистемаЦенообразование.

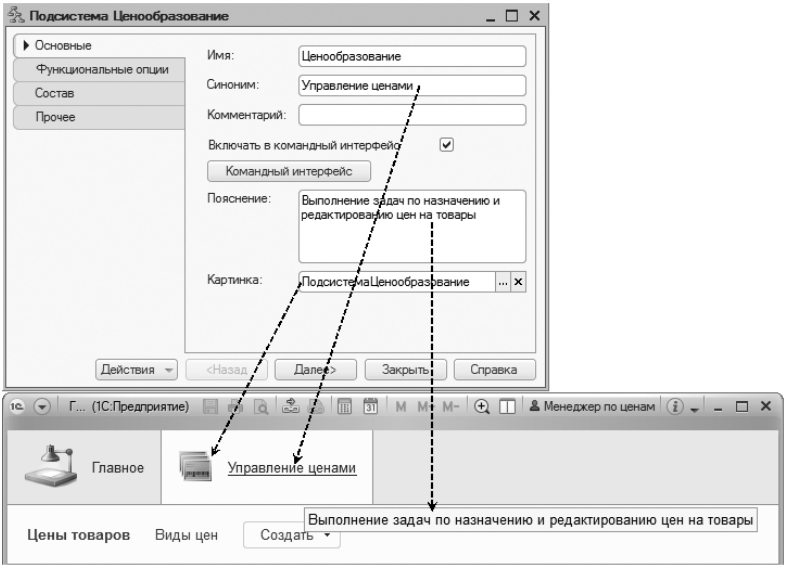


Рис. 1.138. Настройка представления раздела

В результате представление раздела изменилось в соответствии со значениями, заданными для свойств подсистемы.

Глава 1.10. Модель разработки глобального командного интерфейса

Итак, подведем итог и кратко сформулируем основные задачи, решаемые при разработке глобального командного интерфейса.

В общем случае перечень задач, которые необходимо решить при реализации одной итерации цикла разработки, состоит из следующей последовательности:

1. *Проектирование состава подсистем* – структура подсистем имеет одно из важнейших значений, так как именно по ней система будет строить командный интерфейс пользователя, а пользователь – «видеть» прикладное решение.
2. *Формирование состава ролей пользователей* – для пользователей, решающих разные задачи, командный интерфейс должен предоставлять набор команд, соответствующий этой задаче.
3. *Проектирование и настройка функциональных опций* – начиная разработку, следует продумать, какие части функционала должны быть настраиваемыми при внедрении.
4. *Настройка свойств объектов конфигурации* – для объектов конфигурации следует задать свойства, определяющие их принадлежность к подсистемам, функциональным опциям, и состав команд объекта; назначить права доступа по ролям.
5. *Установка интерактивных свойств конфигурации* – для объектов конфигурации следует задать свойства, определяющие их представление в интерфейсе.

Уже эти минимальные действия приведут к тому, что система построит командный интерфейс по умолчанию. В разделах глобального командного интерфейса появятся стандартные команды открытия форм списков объектов, создания новых объектов, вызова отчетов и др.

Стандартные команды объектов, если это определено в их свойствах, также появятся в формах других объектов: команды ввода

на основании, перехода к списку по регистратору, перехода к списку по владельцу и т. д.

Кроме того, для реализации функциональных возможностей прикладного решения разработчик может расширить состав команд путем создания произвольных команд.

Если не устраивает автоматическое расположение и видимость команд, то можно отредактировать фрагменты глобального командного интерфейса и состав команд тех форм, в которых автоматически размещены параметризованные команды объектов (перехода к подчиненному списку и ввода на основании).

При редактировании командного интерфейса необходимо обращать внимание на следующее:

1. *Подбор состава команд* – следует определить команды, которые будут отображаться по умолчанию.
2. *Настройка размещения команд* – следует организовать размещение команд таким образом, чтобы наиболее важные и востребованные команды были всегда под рукой.
3. *Настройка видимости по ролям* – следует продумать, для каких ролей откорректировать видимость по умолчанию, чтобы интерфейс системы не оказался перегружен.

После завершения итерации разработки выполняется *оценка работы интерфейса* – полезно посмотреть, как выглядит интерфейс для типичных пользователей, а также как выглядит интерфейс с выключенными функциональными опциями.

Кроме того, интерфейс приложения адаптируется также и к размерам экранов мобильных устройств. Об этом будет рассказано в пятой части книги «Мобильный клиент» на стр. 867.

Глава 1.11. Создаем произвольные команды

Помимо стандартных команд «1С:Предприятие» позволяет разработчику создавать собственные *произвольные команды* и размещать их в интерфейсе такими же способами, которыми размещаются стандартные команды. Действия, выполняемые этими собственными произвольными командами, разработчик описывает на встроенном языке в специальных модулях – модулях команд.

Произвольные команды

В конфигурации произвольные команды представлены объектом конфигурации Команда. Объект конфигурации Команда предназначен для реализации в прикладном решении *нестандартных функций* с возможностью использования *стандартных механизмов* включения реализованного функционала в командный интерфейс.

Технологическая платформа ничем не ограничивает состав произвольных команд и реализуемые ими функции. Все определяется требованиями к конкретному прикладному решению.

При создании произвольной команды разработчик должен установить ее свойства, определяющие правила включения команды в интерфейс, и написать программный код, определяющий выполняемые командой действия. Этим произвольные команды отличаются от стандартных. Для последних и свойства, и выполняемые действия определены самой платформой.

В конфигурации произвольные команды могут быть реализованы или как независимые объекты – общие команды, или как команды, подчиненные другим объектам.

Список объектов, для которых разработчик может создать подчиненные произвольные команды, приведен в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 6.2.2.

Общие произвольные команды позволяют реализовать нестандартную функциональность, относящуюся в целом к прикладному решению. В этом случае произвольная команда создается как независимый объект конфигурации, принадлежащий классу Общие команды (рис. 1.139).

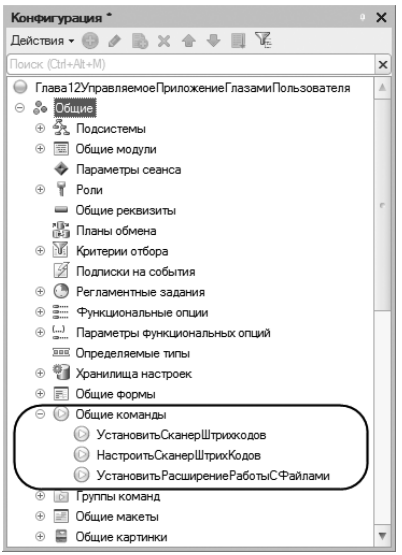


Рис. 1.139. Произвольные общие команды

Подчиненные произвольные команды позволяют реализовать нестандартную функциональность какого-либо объекта конфигурации. В этом случае произвольная команда создается как подчиненный объект конфигурации (рис. 1.140).

Чтобы иметь представление о том, какие задачи решаются с использованием произвольных команд, давайте посмотрим на демонстрационную базу «Глава 1.9. Настраиваем представление команд».

На торговых предприятиях часто требуется автоматизировать процесс регистрации продаваемого товара. Для этого используются сканеры штрихкодов. Однако технологическая платформа ничего «не знает» об этих устройствах и не имеет средств работы с ними. Следовательно, для работы со сканером штрихкодов требуется подключать специальную программу – драйвер. В демонстрационной базе для подключения такого драйвера реализована общая произвольная команда Установить сканер штрихкодов. Установив определенные значения свойств этой команды, разработчик обеспечивает ее доступность пользователям (рис. 1.141).

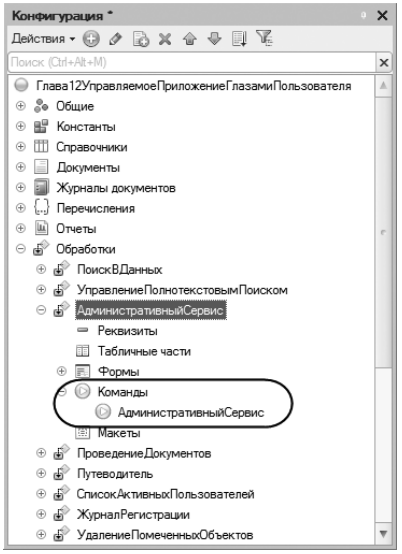


Рис. 1.140. Произвольная подчиненная команда

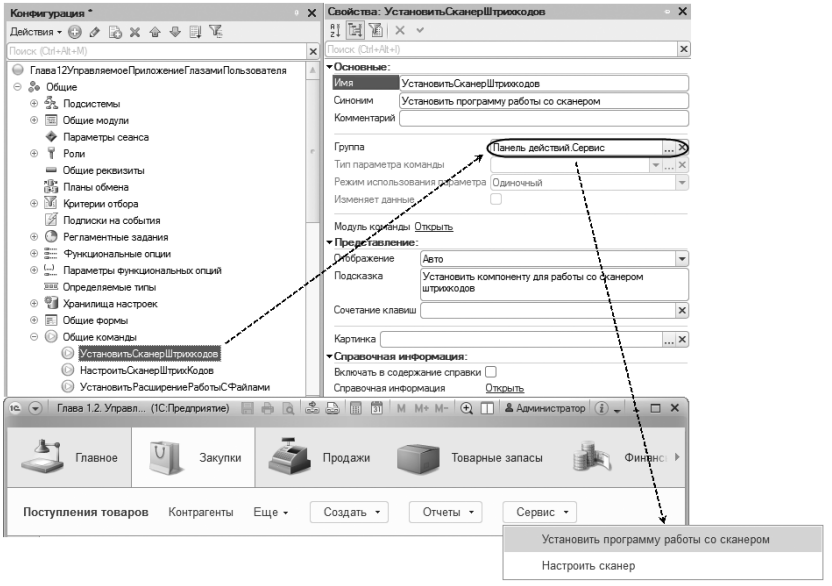


Рис. 1.141. Произвольная общая команда «Установить сканер штрихкодов»

Функции работы со сканером являются общими для всего прикладного решения, то есть не относятся к какому-то конкретному объекту конфигурации, поэтому произвольная команда реализована как общая.

Команда выполняет действие – подключает к прикладному решению драйвер для работы со сканером штрихкодов. Поэтому она расположена в группе Сервис панели функций раздела основного окна приложения.

Еще одной распространенной задачей является получение печатных копий электронных документов. Состав документов, их структура определяются автоматизируемой прикладной задачей. Естественно, в платформе невозможно предусмотреть все многообразие документов и варианты их печатных форм. Для того чтобы «научить» документ «переносить» себя на бумагу, можно воспользоваться произвольной командой.

В демонстрационной базе для получения печатной формы документа РасходТовара создана подчиненная команда ПечатьРасходнойНакладной (рис. 1.142).

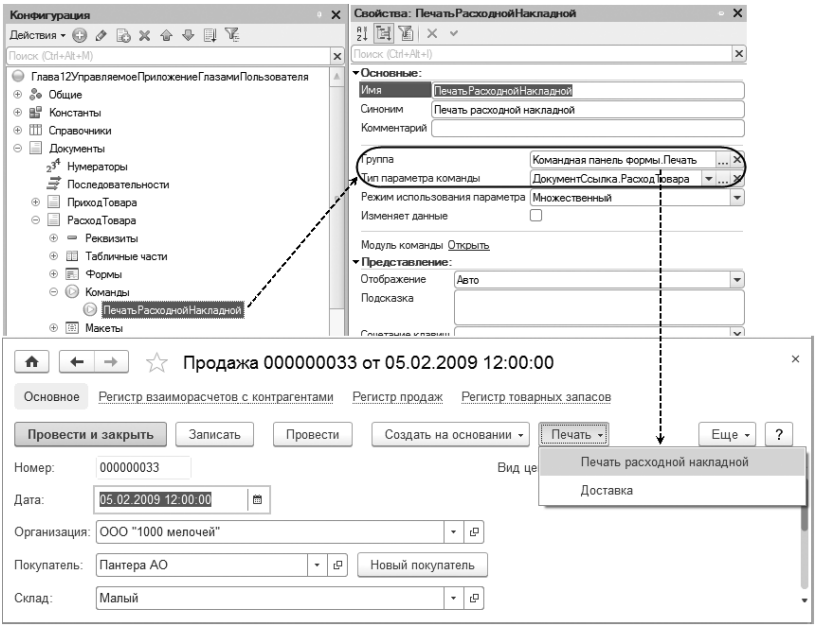


Рис. 1.142. Произвольная подчиненная команда «Печать расходной накладной»

Значения свойств команды Группа и Тип параметра команды определили расположение команды в командном интерфейсе (меню Печать командной панели формы документа), а процедура на встроенном языке обеспечила формирование печатной формы конкретного документа, ссылка на который передается в параметре команды.

Подробнее о размещении произвольных команд рассказано в разделе «Особенности размещения» на стр. 155.

Еще одним, достаточно специфическим, вариантом использования произвольных команд является расширение или переопределение стандартной функциональности стандартной команды. Такие задачи возникают, например, вследствие требования сократить количество ручных операций или изменить стандартное поведение объектов.

Например, в демонстрационной базе реализована обработка АдминистративныйСервис. Команду открытия основной формы необходимо было расположить среди команд навигации. Но стандартная функциональность этого объекта отличается от требуемой – команда открытия формы располагается среди команд действий, в группе Сервис.

Для обеспечения требуемой функциональности у обработки снято свойство Использовать стандартные команды (рис. 1.143) – стандартные команды нас не устраивают.

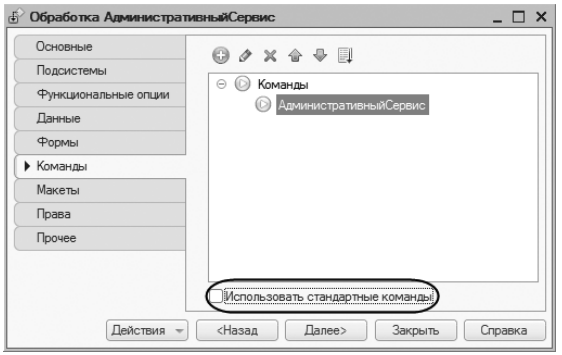


Рис. 1.143. Стандартные команды обработки не используются

Доступ же к обработке обеспечивает произвольная подчиненная команда АдминистративныйСервис, для которой задано расположение в группе Панель навигации.Обычное основного окна приложения (рис. 1.144). В результате выбора этой команды в рабочей области основного окна отображается форма обработки.

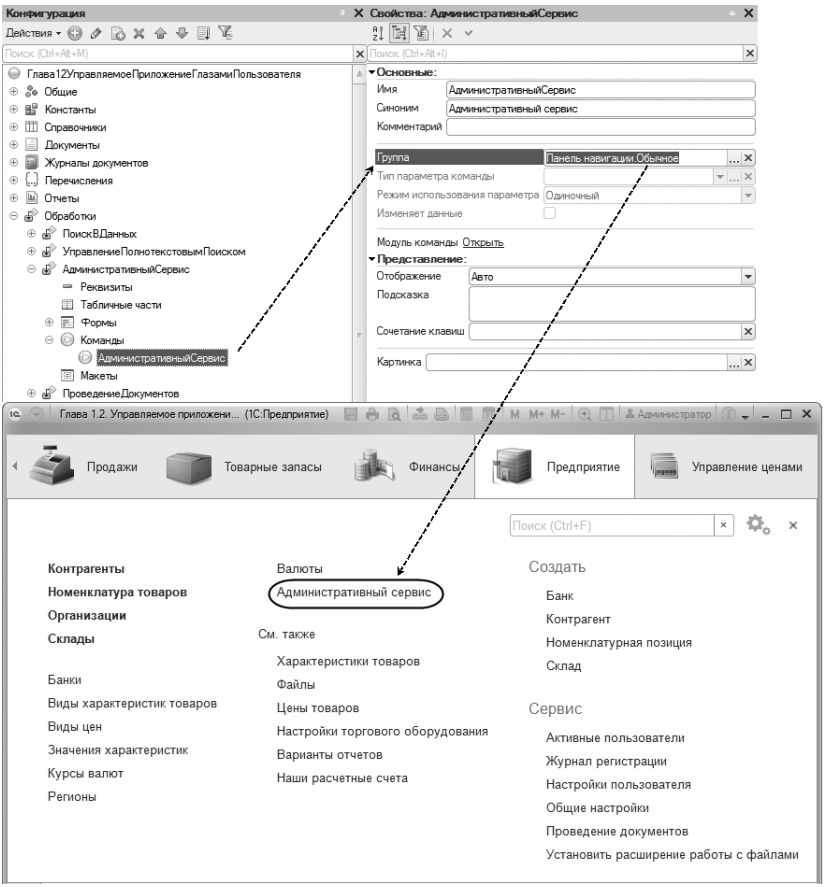


Рис. 1.144. Произвольная подчиненная команда «Административный сервис»

Подобных задач в конкретных прикладных решениях возникает великое множество, и для их решения наиболее подходящими являются произвольные команды.

Особенности размещения

Особенностью произвольных команд по сравнению со стандартными является необходимость описания места их размещения по умолчанию (см. раздел «Категории и группы команд» на стр. 85) в командном интерфейсе. Место размещения произвольной команды задает разработчик при конфигурировании прикладного решения.

Остальные аспекты управления произвольными командами: доступность по ролям (см. раздел «Система прав доступа» на стр. 69), управление видимостью команды (см. раздел «Ручное размещение и видимость команд» на стр. 113), настройка зависимости от функциональных опций (см. раздел «Отключаем неиспользуемые команды» на стр. 119) и т. д. – аналогичны управлению стандартными командами.

Размещение по умолчанию в командном интерфейсе для произвольных команд определяется:

- категорией и группой, назначенными команде;
- принадлежностью команды к подсистеме конфигурации (для независимых команд);
- типом параметра команды (для параметризуемых команд).

Категория команды и ее группа устанавливаются в свойстве Группа этой команды (см. рис. 1.141, 1.142, 1.144).

ВНИМАНИЕ!

Свойство команды Группа обязательно должно быть заполнено. В противном случае возникнет ошибка при обновлении конфигурации базы данных и обновление не выполнится.

При выборе группы для команды следует обращать внимание на необходимость в передаче параметров команде и на действия, выполняемые командой. В качестве общего критерия можно предложить придерживаться тех же правил, которые используются для стандартных команд (см. раздел «Правила размещения глобальных команд» на стр. 93):

- Если команда для своего исполнения не требует параметров, то для нее выбирают группу с категорией Панель навигации или с категорией Панели действий.

- Если команда для своего исполнения требует передачи параметра, то для нее необходимо выбрать группу с категорией Панель навигации формы или с категорией Командная панель формы.
- Для команд, выполнение которых приводит к изменению информации, отображаемой в рабочей области того же окна, следует выбирать категорию Панель навигации для независимых команд или Панель навигации формы для параметризуемых команд.
- Для команд, выполнение которых приводит к изменению данных в информационной базе, следует выбирать категорию Панель действий для независимых команд или Командная панель формы для параметризуемых команд. Также эту категорию рекомендуется выбирать для команд, которые приводят к открытию блокирующих окон.

Для включения *общей независимой команды* в тот или иной раздел командного интерфейса необходимо указать ее принадлежность к соответствующим подсистемам. Включение команды в подсистемы выполняется путем ее отметки в свойстве Состав требуемых подсистем (см. раздел «Стандартные команды» на стр. 60).

Подчиненную же команду непосредственно включить в подсистему невозможно. Поэтому *подчиненные независимые команды* автоматически включаются в командный интерфейс тех подсистем, в которые включен объект – владелец команды.

Именно поэтому параметризуемые команды можно размещать только в панели навигации формы или в командной панели формы.

А вот параметризуемые произвольные команды, как общие, так и подчиненные, включаются в командный интерфейс иначе. Связано это с тем, что фактическое значение своего параметра команда может получить только из данных формы.

Причем это значение должно иметь тип данных, допустимый для параметра. Состав допустимых типов параметра устанавливается в свойстве Тип параметра команды (рис. 1.145).

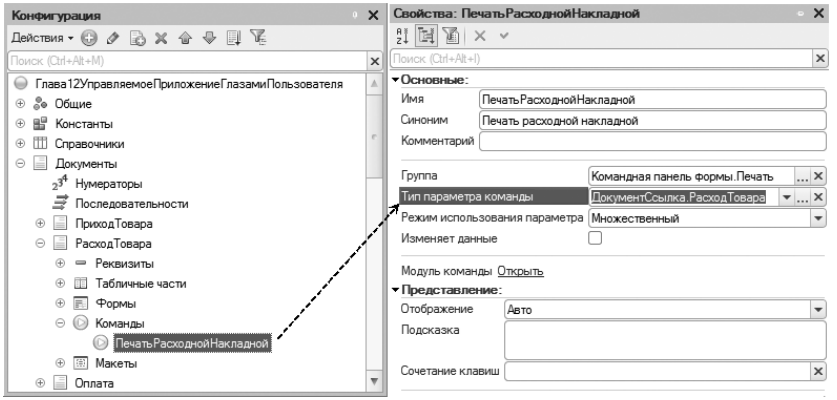


Рис. 1.145. Состав допустимых типов параметров для параметризуемой команды определяется ее свойством «Тип параметра команды»

Сравнивая состав типов, заданных в свойстве команды, с типами реквизитов формы, система принимает решение о включении команды в ту или иную форму.

Параметризуемая произвольная команда включается в форму только тогда, когда форма имеет хотя бы один реквизит с типом, входящим в состав допустимых. При проверке учитываются и реквизиты, подчиненные основному реквизиту формы. Состав проверяемых подчиненных реквизитов ограничен первым уровнем подчинения.

Развитие функциональности ценообразования

Вернемся к решению нашей задачи по выделению функционала работы с ценами в отдельную подсистему. Кроме назначения цен нам потребуется и возможность распечатывать ценники на товары. При этом необходимо реализовать два режима:

- печать ценников на все товары по всем существующим видам цен,
- печать ценников на все товары по одному виду цен.

Стандартные команды объектов конфигурации не могут обеспечить нас этими возможностями. Следовательно, нам необходимо

реализовать дополнительную функциональность. Для этого будем использовать произвольные команды.

В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.9. Настраиваем представление команд». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.11. Создаем произвольные команды».

Общая независимая команда

Функционал печати всех ценников, на первый взгляд, должен расширять возможности справочника Товары. Однако этот объект конфигурации описывает *множество* объектов данных информационной базы. Если мы реализуем команду как подчиненную справочнику, то мы «научим» каждый из объектов данных печатать ценники на все товары. А это уже лишнее. Объект данных должен быть ответственен только за себя. Поэтому команда будет общей.

Используя контекстное меню узла Общие команды дерева конфигурации, добавим новую команду. Для новой команды откроются палитра свойств и окно редактирования модуля команды с шаблоном обработчика команды.

В группе свойств Основные зададим значения свойств команды (рис. 1.146):

- Имя – ПечатьЦенниковТовары;
- Синоним – оставим автоматически сформированный синоним;
- Комментарий – заполнять не будем.

Следующий шаг – выбор категории команды и группы для ее размещения по умолчанию.

Наша команда для своего исполнения не требует параметров – она *независимая*. Команда выполняет *действия* по обработке данных, хранимых в информационной базе, с целью получения набора ценников, а не изменяет контекст решения какой-либо задачи. Следовательно, для команды установим категорию Панель действий. А в какой группе она будет отображаться? Логичнее всего поместить ее в группу Сервис.

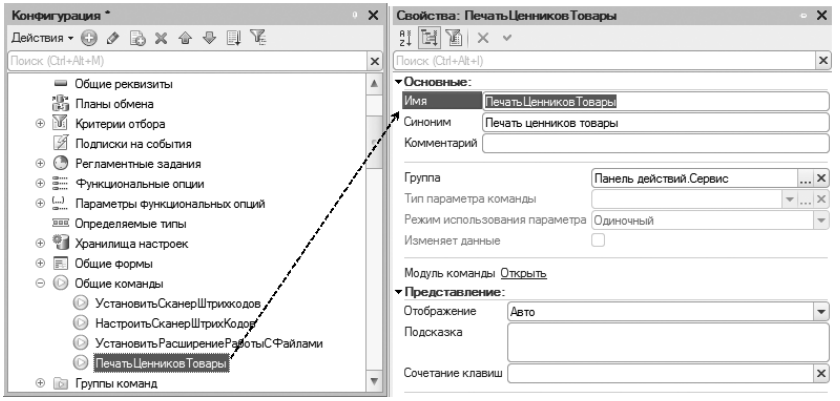


Рис. 1.146. Свойства общей произвольной команды

Поэтому для свойства Группа открываем окно со списком групп и выбираем элемент ПанельДействий.Сервис (рис. 1.147).

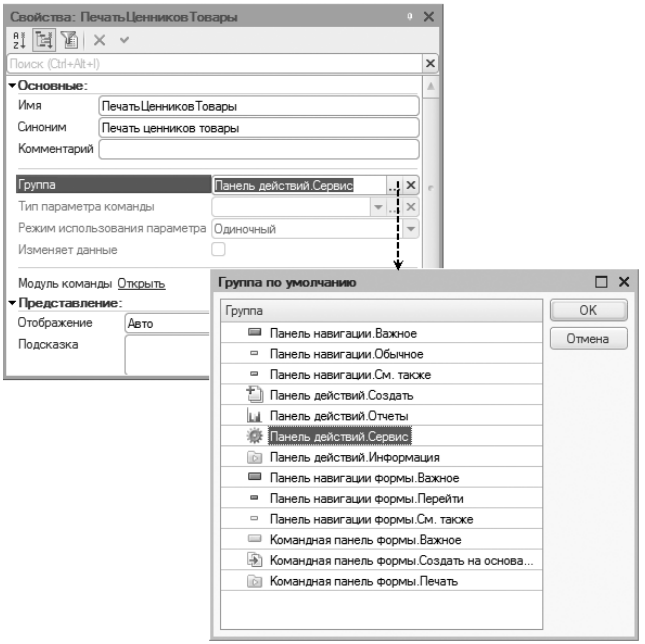


Рис. 1.147. Задание места размещения произвольной общей команды

ПРИМЕЧАНИЕ

Обратите внимание на свойства Тип параметра команды, Режим использования параметра и Изменяет данные – они недоступны для заполнения. Свойства предназначены для описания параметризуемой команды и становятся доступны только при выборе группы с категориями Панель навигации формы или Командная панель формы.

Созданная нами команда является независимой. Следовательно, необходимо определить, в каких разделах командного интерфейса она будет доступна. Команда должна быть доступна в тех же разделах, в которых можно выполнять действия по управлению ценами.

В нашем случае созданная команда должна принадлежать трем подсистемам: Ценообразование, Цены, Предприятие. Таким образом, нам требуется отредактировать свойство Состав трех подсистем.

Для сокращения количества выполняемых действий из контекстного меню созданной команды выберем пункт Дополнительно. В результате откроется окно, в котором на закладке Подсистемы можно указать все подсистемы, которым принадлежит команда (рис. 1.148).

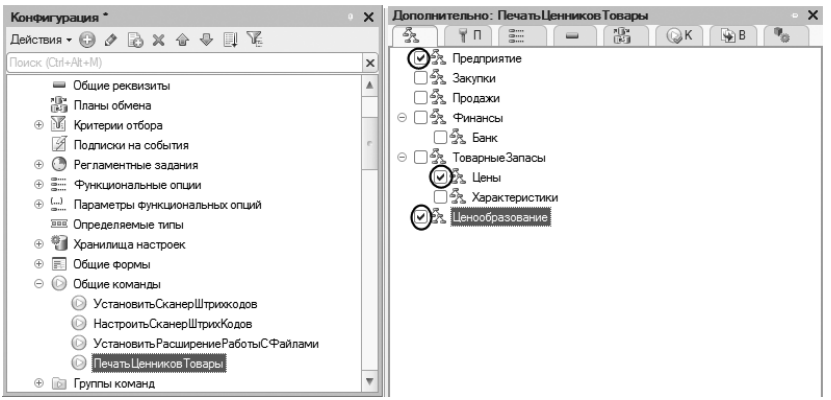


Рис. 1.148. Включение произвольной общей команды в подсистемы

Следующая задача – назначение прав на использование команды. Наша команда доступна пользователям с ролью Администратор

за счет установленного свойства роли Устанавливать права для новых объектов. От нас требуется обеспечить ее доступность и для роли Менеджер по ценам.

Как и для других объектов конфигурации, для общей команды настройку доступности можно выполнить в окне редактирования роли (см. раздел «Система прав доступа» на стр. 69). А можно в уже открытом окне Дополнительно на закладке Права. В списке Роли выбираем настраиваемую роль и в списке Права устанавливаем право Просмотр для созданной команды (рис. 1.149).

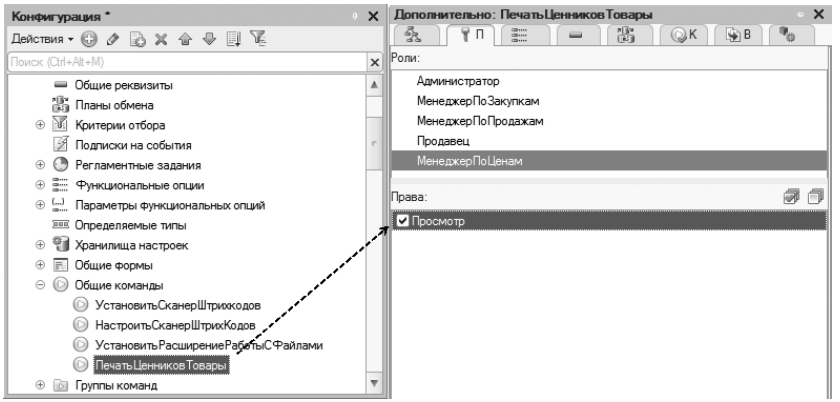


Рис. 1.149. Настройка доступности команды для роли «Менеджер по ценам»

Настройка видимости команды по ролям для произвольной независимой команды выполняется аналогично настройке для стандартных команд – в редакторе командного интерфейса (см. раздел «Система настройки командного интерфейса» на стр. 97).

Наша команда по умолчанию должна быть видима пользователю с ролью Менеджер по ценам, а от пользователя с ролью Администратор ее необходимо скрыть. Для этого в редакторе командного интерфейса подсистемы Ценообразование снимем флажок общей видимости в колонке Видимость. Это обеспечит нам невидимость команды для всех ролей, в том числе и вновь создаваемых. А для роли Менеджер по ценам явным образом установим флажок в соответствующей колонке (рис. 1.150).

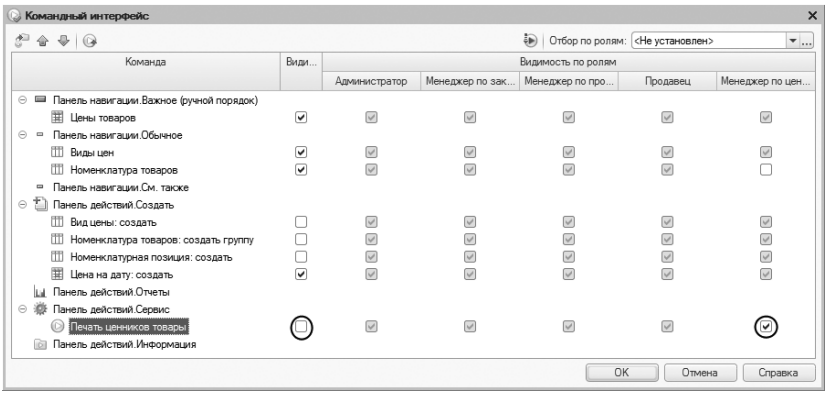


Рис. 1.150. Настройка видимости произвольной общей команды

Сохраним конфигурацию, запустим приложение от имени пользователя Менеджер по ценам и выберем раздел Управление ценами.

В командном интерфейсе (рис. 1.151) команда Печать ценников товары доступна в разделе Управление ценами (за счет указания принадлежности к подсистеме Ценообразование). Команда размещена в группе Сервис панели действий (за счет указания соответствующего значения свойства Группа).

Таким образом, для произвольной общей независимой команды:

- размещение в командном интерфейсе по умолчанию определяется значением свойства Группа;
- включение в раздел командного интерфейса определяется принадлежностью к соответствующей подсистеме;
- доступность для пользователя определяется значением права Просмотр.

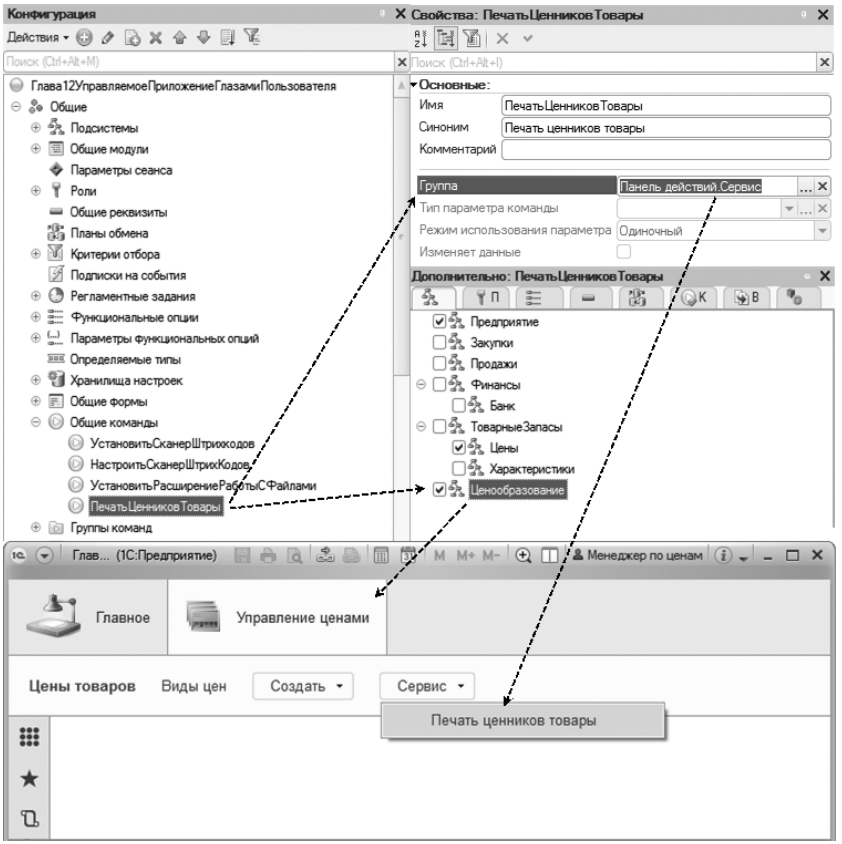


Рис. 1.151. Размещение произвольной общей независимой команды в командном интерфейсе

Команду мы создали. А как рассказать пользователю о том, какие действия выполняет произвольная команда? Ответ очевиден – описать назначение команды в документации к прикладному решению. Также назначение команды можно описать во встроенной электронной справке. Для работы со справочной информацией предназначены свойства общей команды из группы Справочная информация. Вызовем соответствующий пункт из контекстного меню команды и в открывшемся окне введем поясняющий текст (рис. 1.152).

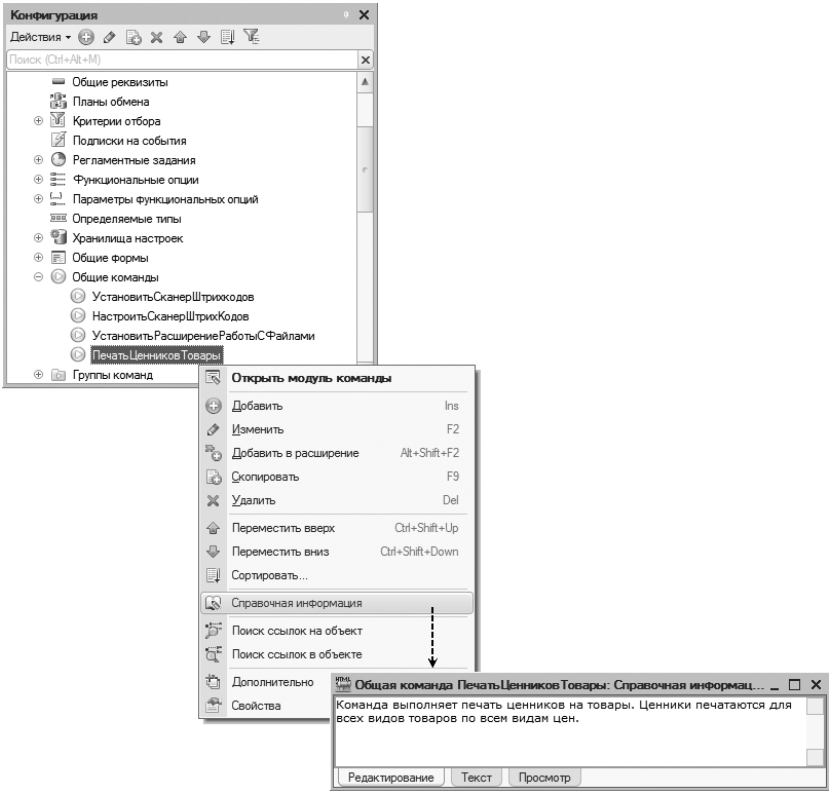


Рис. 1.152. Справочная информация по произвольной общей команде «ПечатьЦенниковТовары»

Однако поиск описания команды в документации или встроенной справке – процесс длительный. К тому же, чтобы увидеть справочную информацию, пользователю нужно будет лишний раз нажать кнопку со знаком вопроса. Можно помочь пользователю быстро вспомнить назначение команды, выбрав для нее говорящее представление.

Произвольная общая команда в командном интерфейсе представляется своим свойством Синоним. Сейчас команда представлена текстом «Печать ценников товары», и это представление достаточно информативно. Но в дальнейшем мы добавим в прикладное решение еще одну команду печати ценников – по виду цен. Поэтому стоит

подумать над таким представлением команды, которое подскажет пользователю, какой вариант печати ценников будет выполнен. Скажем, это будет «Печать всех ценников».

Еще одним способом напоминания пользователю о назначении команды является использование свойства Подсказка. Текст, заданный в этом свойстве, отображается во всплывающей подсказке при наведении указателя мыши на команду. Для свойства Подсказка зададим текст «Печать ценников на все товары по всем видам цен».

В результате изменения значений свойств Синоним и Подсказка представление команды в командном интерфейсе изменится следующим образом (рис. 1.153).

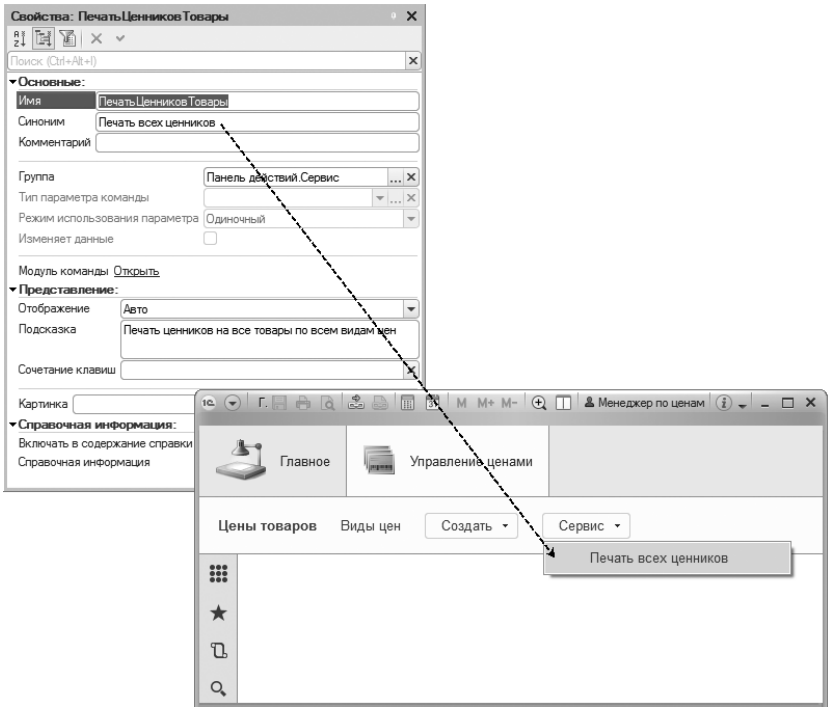


Рис. 1.153. Измененное представление команды

На данный момент мы создали команду, настроили ее расположение, доступность и представление. Нам осталось реализовать функциональность команды, то есть те действия, которые должна выполнять команда.

Для определения выполняемых действий разработчику необходимо реализовать процедуру на встроенном языке. Мы не будем реализовывать требуемую функциональность команды. Для нас достаточно просто увидеть реакцию системы на выбор команды. Поэтому в модуле команды просто выведем сообщение (листинг 1.1):

Листинг 1.1. Обработчик команды «ПечатьЦенниковТовары»

```
Сообщение = Новый СообщениеПользователю();
Сообщение.Текст = "Необходимо реализовать алгоритм выполнения команды";
Сообщение.Сообщить();
```

Чтобы открыть модуль команды, достаточно дважды щелкнуть на имени команды в дереве объектов конфигурации. При создании команды в модуле был автоматически создан ее обработчик. Заполним обработчик текстом, приведенным в листинге 1.1 (рис. 1.154).

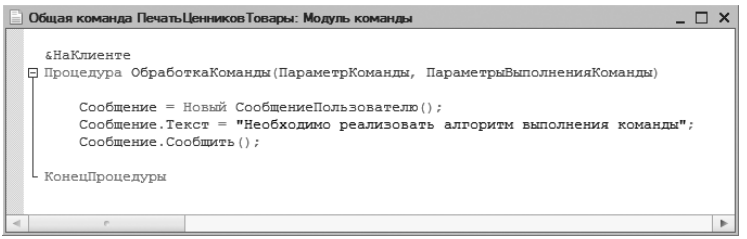


Рис. 1.154. Процедура, реализующая функциональность произвольной команды

При выборе команды в режиме 1С:Предприятие в окне сообщений будет выведен текст нашего сообщения (рис. 1.155).

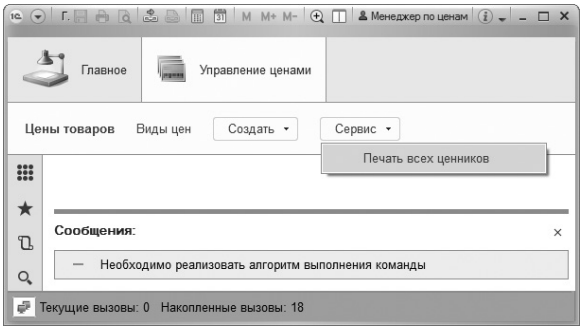


Рис. 1.155. Результат исполнения произвольной общей команды

Подчиненная параметризуемая команда

Теперь реализуем функциональность печати ценников по одному виду цен. Для этого также будем использовать произвольную команду.

В отличие от предыдущей эта команда расширяет функциональные возможности справочника Виды цен – любой из элементов справочника (объектов, хранимых в информационной базе) должен знать, как распечатать «свои» ценники (по виду цены). Поэтому команду реализуем как подчиненный справочнику Виды цен объект Команда.

Для добавления произвольной подчиненной команды выберем пункт Добавить контекстного меню узла Команды справочника Виды цен. Для новой команды откроются палитра свойств и окно редактирования модуля команды с шаблоном обработчика команды.

Состав свойств подчиненной произвольной команды практически полностью совпадает со свойствами общей. Особенностью является отсутствие у подчиненной команды группы свойств Справочная информация – описание команды включается в справку самого объекта-владельца.

В группе свойств Основные заполним следующие свойства (рис. 1.156):

- Имя – ПечатьЦенниковВидЦены;
- Синоним – оставим автоматически сформированный синоним;
- Комментарий – заполнять не будем.

Для свойства Группа пока установим значение ПанельДействий.Сервис – этим мы определили произвольную подчиненную *независимую* команду *действия*.

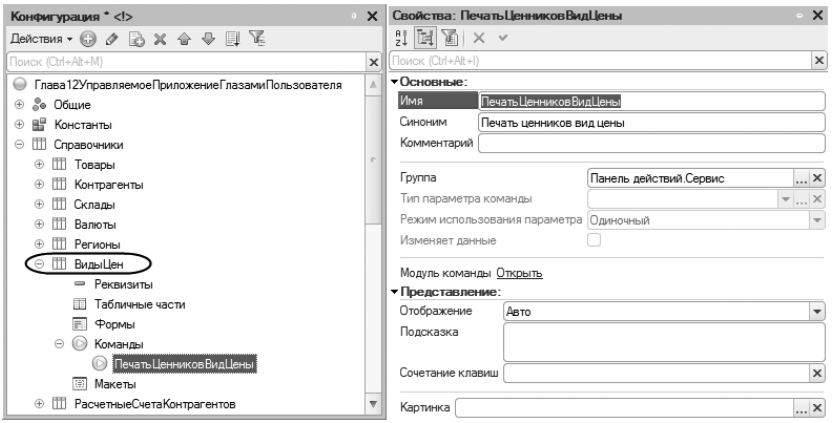


Рис. 1.156. Свойства произвольной подчиненной команды

Подчиненная команда включается в те же разделы интерфейса, что и объект-владелец. Следовательно, созданная команда включена в раздел Предприятие, раздел Управление ценами и подраздел Цены раздела Товарные запасы за счет принадлежности справочника Виды цен соответствующим подсистемам (рис. 1.157).

Теперь давайте вспомним, что команда выполняет *действия* по печати ценников только для выбранного вида цен. Этот вид цены команда принимает как управляющий параметр, то есть команда *параметризуемая*.

Поэтому, ориентируясь на общий критерий выбора категории, изменим у команды значение свойства Группа на Командная панель формы. Важное – команда по умолчанию располагается в группе Важное командной панели формы.

Обратите внимание, что стали доступны для заполнения свойства Тип параметра команды, Режим использования параметра и Изменяет данные.

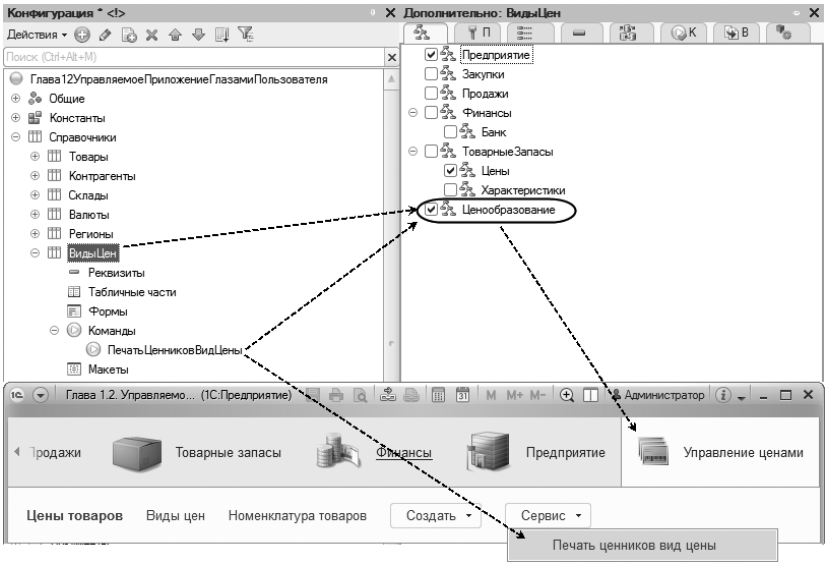


Рис. 1.157. Принадлежность к подсистеме для произвольной подчиненной команды определяется объектом-владельцем

В качестве параметра наша команда принимает ссылку на тот вид цен (элемент справочника Виды цен), для которого необходимо распечатать ценники. Поэтому в качестве значения свойства Тип параметра команды зададим тип данных СправочникСсылка.ВидыЦен (рис. 1.158).

В случае если параметр может принимать значения разных типов, для свойства Тип параметра команды необходимо выбрать составной тип данных.

Исходя из типа параметра, наша команда будет доступна как минимум в двух формах:

- в форме списка справочника Виды цен, где параметром будет ссылка на текущую строку списка;
- в форме элемента того же справочника, параметром будет ссылка на редактируемый объект.

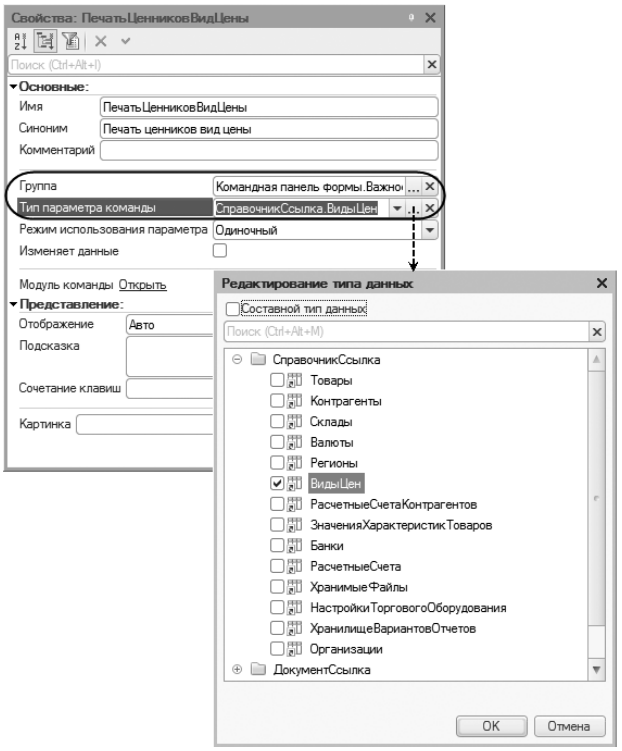


Рис. 1.158. Группа и тип параметра определяют расположение произвольной параметризуемой команды

А если команде потребуется передать не одно значение параметра, а несколько? В этом случае необходимо воспользоваться свойством Режим использования параметра. Это свойство предназначено для определения количества значений, которые будут переданы в качестве параметра (рис. 1.159).

Если свойство установлено в значение Одиночный, то в команду передается одно значение указанного типа.

Если свойство установлено в значение Множественный, то в команду всегда передается массив значений указанного типа. Этот режим имеет смысл выбирать тогда, когда источником параметра является таблица с установленным режимом множественного выделения.

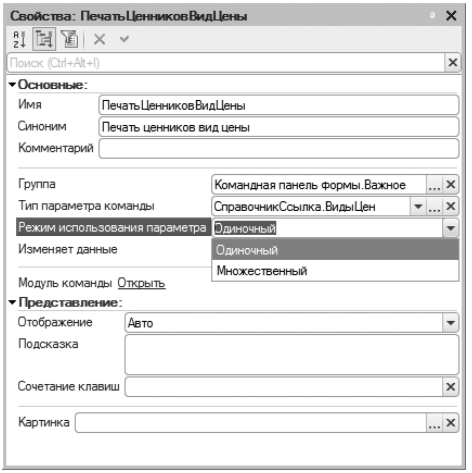


Рис. 1.159. Возможность использования многозначного параметра определяется свойством «Режим использования параметра»

Если в таблице выделено несколько строк, то первым элементом массива будет выступать текущая строка (вне зависимости от последовательности выделения строк табличного поля). Порядок элементов после первого не определен. Если же текущая строка не входит в выделение, то первый элемент в массиве невозможно предсказать однозначно.

Массив значений будет передан и в том случае, если в таблице выделена только одна строка. Такой массив будет содержать только один элемент.

Еще одним свойством, которое определено для параметризуемых команд, является свойство Изменяет данные. Если это свойство установлено, то система пытается заблокировать данные формы для редактирования. В случае успеха для формы устанавливается признак модифицированности. Такое поведение позволяет сымитировать интерактивное изменение данных в форме.

Как и для команды печати всех ценников, для нашей команды необходимо выполнить назначение прав на ее использование. Команда должна быть доступна пользователям с ролью Менеджер по ценам.

Для настройки доступности воспользуемся установкой прав на подчиненные объекты (см. раздел «Система прав доступа» на стр. 65). Для этого в контекстном меню подчиненной команды выберем пункт Дополнительно. В открывшемся окне на закладке Права установим право Просмотр для роли Менеджер по ценам (рис. 1.160).

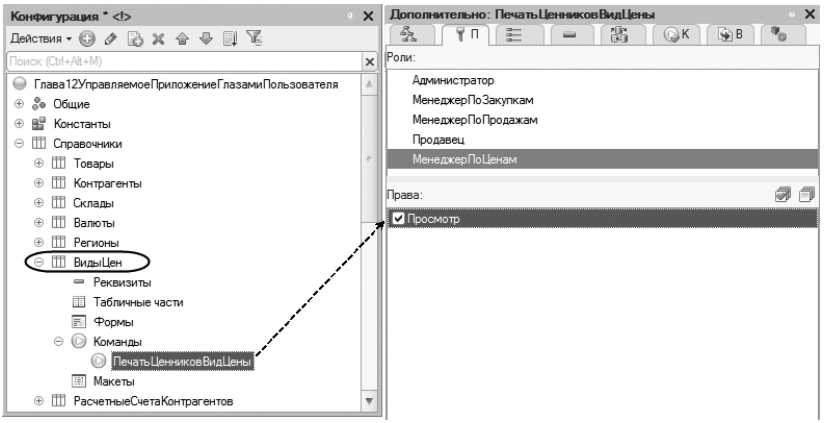


Рис. 1.160. Ролевая доступность произвольной подчиненной команды определяется ее правом «Просмотр»

Если право Просмотр будет сброшено, система не включит подчиненную команду в командный интерфейс пользователя. Даже в том случае, когда роль обладает полными правами на объект-владелец (установлены все права на этот объект).

Настройка видимости параметризуемой команды по ролям для произвольной параметризуемой команды выполняется в редакторе формы. Об этом мы поговорим в главе 1.12 (стр. 186).

В результате выполненных настроек команда печати ценников для вида цены доступна в командных панелях формы списка справочника Виды цен и в форме элемента того же справочника для пользователей с ролью Менеджер по ценам.

Сохраним конфигурацию, запустим приложение от имени пользователя Менеджер по ценам, выберем раздел Управление ценами и откроем форму списка справочника Виды цен. Созданная нами

команда действительно расположена в группе Важное командной панели формы (рис. 1.161).

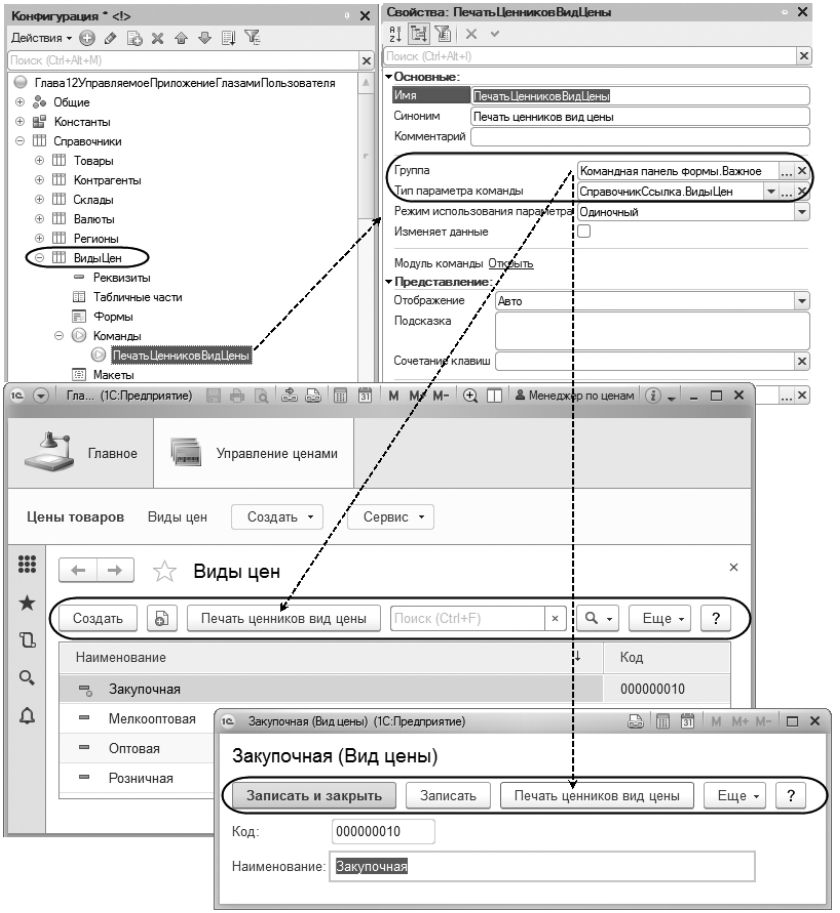


Рис. 1.161. Размещение произвольной параметризуемой подчиненной команды в командном интерфейсе

Таким образом, для произвольной подчиненной параметризуемой команды:

- размещение в командном интерфейсе определяется значением свойства Группа;

- состав форм, в которых будет доступна команда, определяется значением свойства Тип параметра команды;
- доступность для пользователя определяется значением права Просмотр, устанавливаемого для подчиненной команды.

Нам осталось настроить представление команды в интерфейсе и реализовать процедуру исполнения команды.

Для настройки представления заполним свойство Синоним текстом «Печать ценников для вида цены», свойство Подсказка – текстом «Печать ценников на товары по выбранному виду цены» (рис. 1.162).

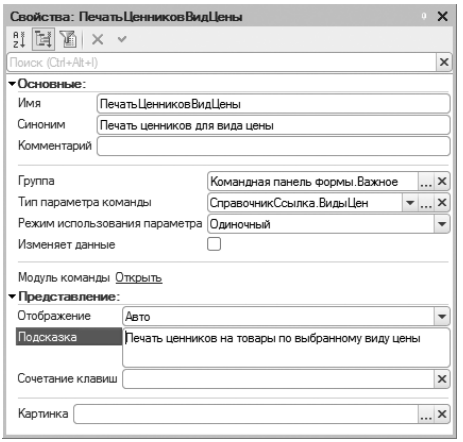


Рис. 1.162. Настройка представления произвольной подчиненной команды

Для того чтобы видеть реакцию на выбор команды, в модуле команды реализуем вывод сообщения (листинг 1.2):

Листинг 1.2. Обработчик команды «ПечатьЦенниковВидЦены»

```
Сообщение = Новый СообщениеПользователю();
Сообщение.Текст = "Необходимо реализовать алгоритм выполнения команды";
Сообщение.Сообщить();
```

В результате в режиме 1С:Предприятие при наведении мыши на кнопку Печать ценников для вида цены всплывает поясняющая подсказка. А при нажатии на эту кнопку в окне сообщений будет выведен текст нашего сообщения (рис. 1.163).

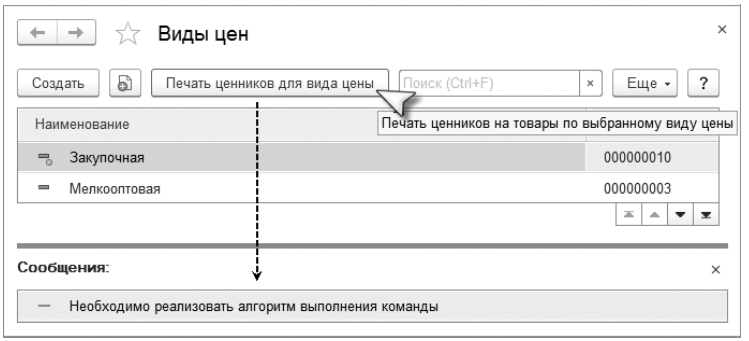


Рис. 1.163. Результат исполнения произвольной подчиненной команды

Помимо свойств Синоним и Подсказка на представление команды в командном интерфейсе также влияют свойства Картинка и Отображение.

Свойства Картинка и Отображение определены и для общих произвольных команд.

На данный момент свойство Картинка не заполнено, а свойство Отображение имеет значение Авто. В этом случае команда представлена значением своего свойства Синоним (см. рис. 1.163).

Свойство Картинка содержит пиктограмму, которая, в зависимости от значения свойства Отображение, может представлять команду в интерфейсе. В качестве пиктограммы может быть выбрана любая стандартная или общая картинка. Давайте для команды зададим стандартную картинку Печать (рис. 1.164).

Свойство Отображение задает способ отображения команды:

- Текст – в представлении команды присутствует только текст.
- Картинка – в представлении команды присутствует только пиктограмма.
- Картинка и текст – в представлении команды присутствуют и текст, и пиктограмма.
- Авто – решение о представлении команды принимает система (в командных панелях, связанных с интерфейсными элементами, присутствуют пиктограмма; в пунктах меню присутствуют текст и пиктограмма).

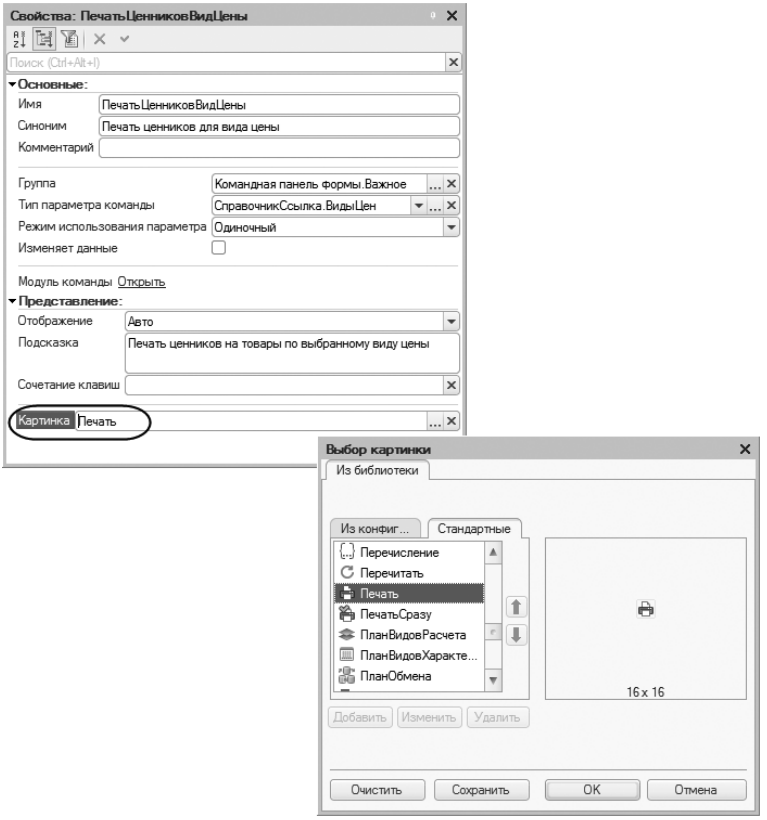


Рис. 1.164. Назначение картинки для произвольной подчиненной команды

Установим для свойства значение Картинка и текст. В результате команда представлена и пиктограммой, и текстом (рис. 1.165).

Свойства Картинка и Отображение используются только для команд, размещенных в командной панели формы. При размещении в других панелях команды представляются своим синонимом (или именем, если синоним не заполнен).

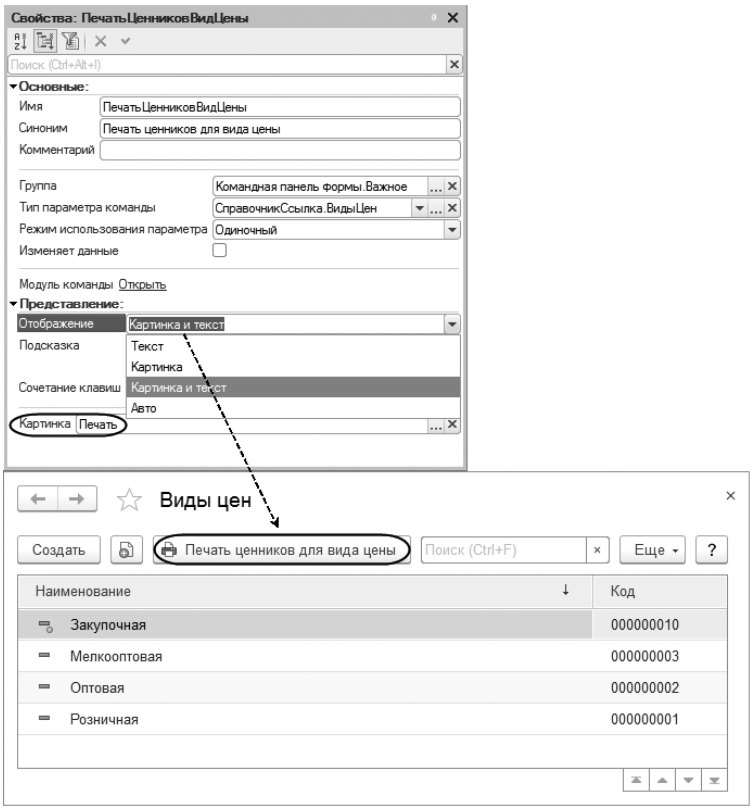


Рис. 1.165. Измененное представление произвольной подчиненной команды

У произвольных команд есть еще свойство, которое мы не рассмотрели, – Сочетание клавиш. Это свойство позволяет назначить команде клавиатурную комбинацию для ее быстрого вызова. Для назначения клавиатурной комбинации необходимо выбрать свойство Сочетание клавиш и нажать на клавиатуре требуемую комбинацию клавиш.

Назначим нашей команде печати ценников по виду цен комбинацию клавиш Ctrl + P. После этого команду можно будет быстро выполнить, нажав указанную комбинацию клавиш (рис. 1.166).

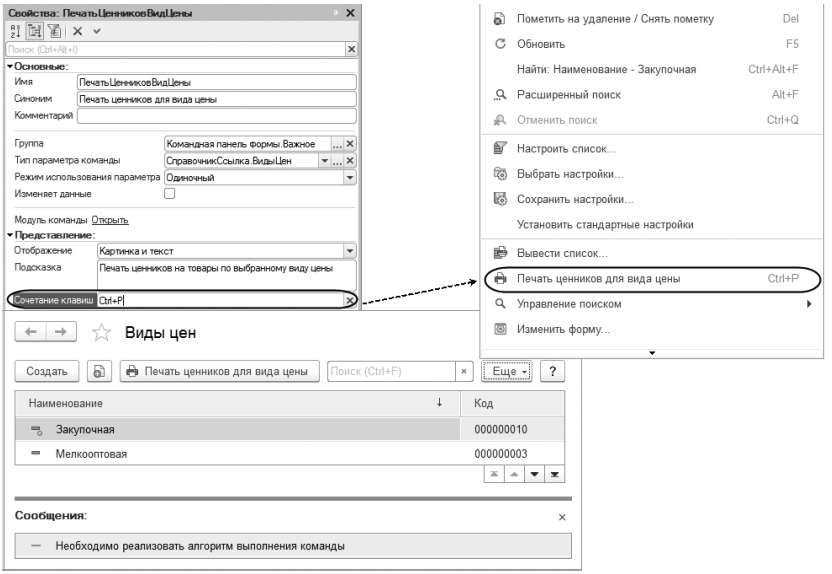


Рис. 1.166. Произвольная подчиненная команда с назначенной клавиатурной комбинацией

Зависимость от функциональных опций

А теперь вспомним о том, что ценообразование в разрезе видов цен является опциональным. Если цены назначаются только в разрезе товаров, то печать ценников в разрезе видов цен не потребуется. Поэтому доступность команды печати ценников по виду цен поставим в зависимость от значения функциональной опции ЦенообразованиеПоВидамЦен.

Для настройки зависимости произвольной команды от функциональной опции необходимо эту команду включить в свойство Состав управляющей функциональной опции (см. раздел «Отключаем неиспользуемые команды» на стр. 119). Как и для включения в подсистемы, для включения команды в функциональные опции воспользуемся окном Дополнительно, вызванным из контекстного меню для подчиненной команды (рис. 1.167).

Аналогично можно настроить зависимость от функциональных опций и для общей произвольной команды.

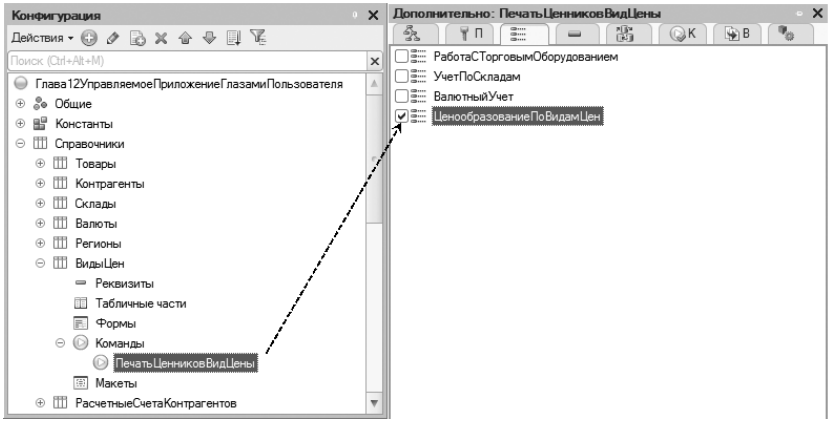


Рис. 1.167. Настройка зависимости произвольной команды от функциональной опции

Также для нашей команды в свойстве Отображение установим значение Картинка.

Сохраним конфигурацию, запустим приложение от имени пользователя Администратор, выполним команду Общие настройки из группы Сервис раздела Предприятие и убедимся, что значение константы Ценообразование по видам цен установлено (рис. 1.168).

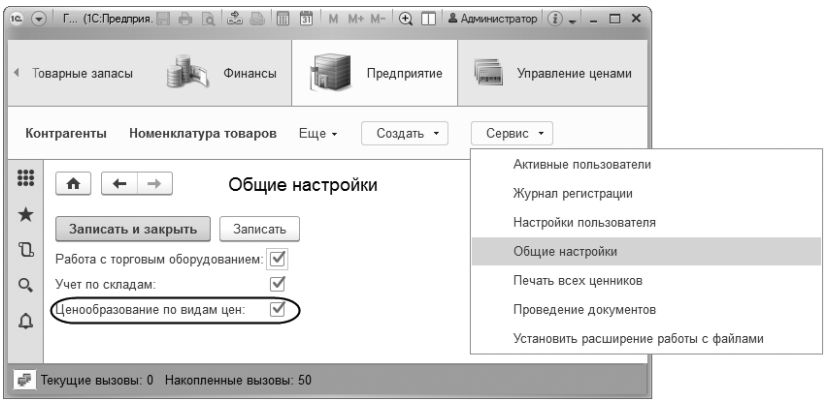


Рис. 1.168. Установка ценообразования в разрезе видов цен

Теперь запустим приложение от имени пользователя Менеджер по ценам и откроем форму списка справочника Виды цен. Команда печати доступна в командной панели формы и отображается только пиктограммой (рис. 1.169).

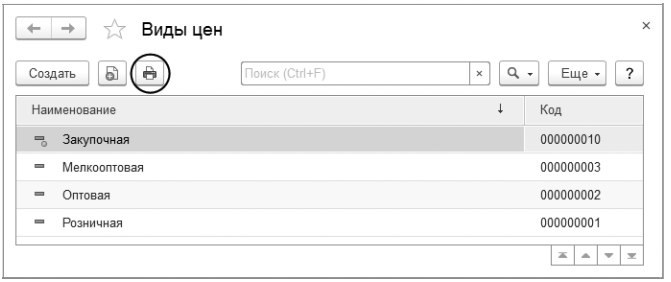


Рис. 1.169. При истинном значении функциональной опции произвольная команда доступна

Изменим значение константы Ценообразование по видам цен на Ложь, перезапустим приложение от имени пользователя с ролью Администратор и откроем форму списка справочника Виды цен через меню Все функции. Команда печати недоступна.

Поскольку сам справочник Виды цен также отключен функциональной опцией, форма списка справочника была открыта через пункт Все функции... главного меню.

Произвольные группы

В разделе «Автоматическое размещение и видимость команд» (стр. 83) мы говорили о том, что команды располагаются в стандартных группах и что при необходимости разработчик может расширить стандартный состав групп. Для этого используются объекты конфигурации Группа команд, расположенные в ветке Общие дерева конфигурации.

Произвольная группа команд используется для логического объединения команд, выполняющих похожие действия.

Например, в демонстрационной базе создана произвольная группа команд Печать (рис. 1.170), предназначенная для объединения команд, которые формируют различные печатные формы.

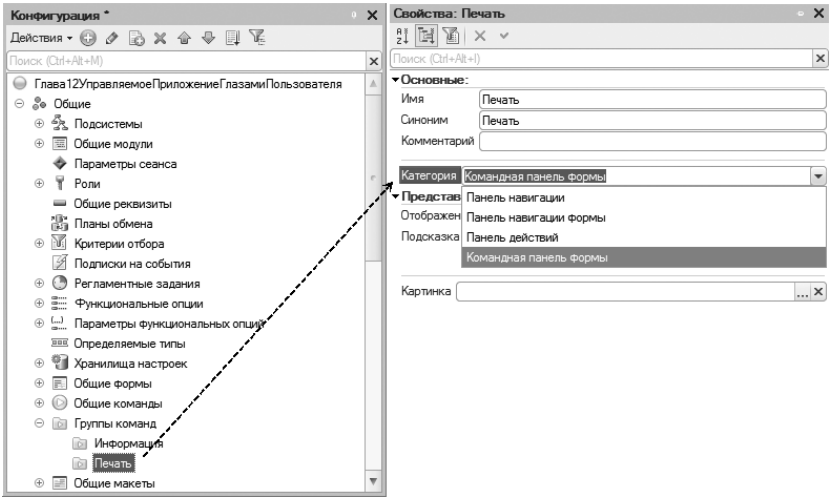


Рис. 1.170. Произвольная группа команд «Печать»

Как и для других объектов конфигурации, для объекта Группа команд определены универсальные свойства Имя, Синоним, Комментарий.

Для определения места размещения группы используется свойство Категория.

ВНИМАНИЕ!

Свойство группы Категория обязательно должно быть заполнено. Если значение для свойства не выбрано, система установит значение по умолчанию Панель навигации.

Выбранная категория определяет, где будет размещена группа, и, следовательно, команды, принадлежащие этой группе. В качестве значения может быть выбрана только одна из стандартных категорий команд (см. раздел «Категории и группы команд» на стр. 85).

Для группы команд Печать установлена категория Командная панель формы. Это значит, что в интерфейсе произвольная группа размещена в командной панели формы, в которой может быть показана произвольная параметризированная команда (рис. 1.171).

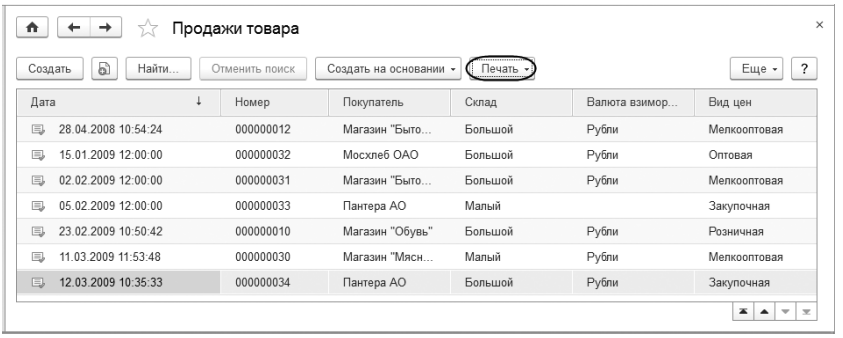


Рис. 1.171. Свойство «Категория» определяет размещение произвольной группы

Давайте включим произвольную команду печати ценников по виду цен в группу Печать. Для этого в свойстве Группа нашей подчиненной команды установим произвольную группу Командная панель формы.Печать.

Кроме того, для свойства Отображение установим значение Авто (рис. 1.172).

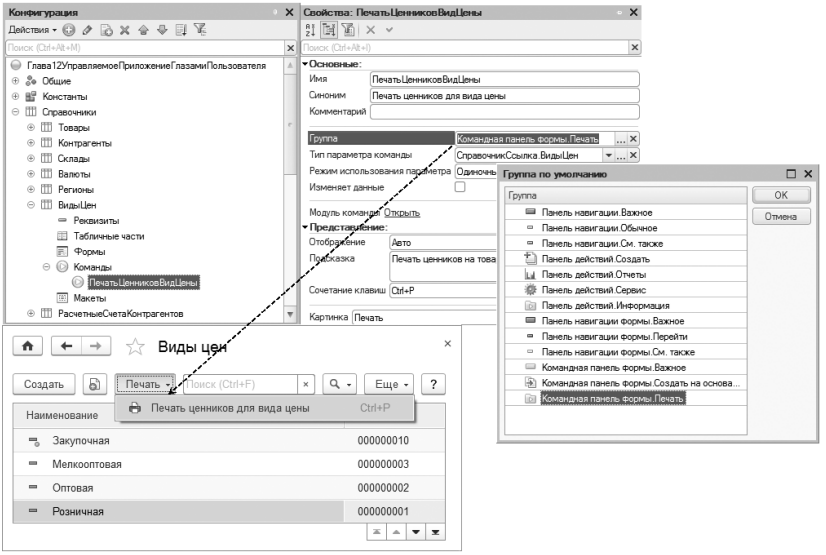


Рис. 1.172. Включение произвольной команды в произвольную группу

Теперь в форме списка и форме элемента справочника Виды цен команда печати ценников располагается уже не в группе Важное, а в группе Печать.

Для формирования представления группы в командном интерфейсе используются свойства Синоним, Отображение, Подсказка и Картинка.

В командной панели формы кнопками отображаются команды из группы Важное. Команды остальных групп отображаются в виде подменю.

Свойство Синоним содержит текст, представляющий произвольную группу в командном интерфейсе. Для синонима зададим значение «Печатные формы».

Свойство Отображение содержит вариант отображения группы команд. Для отображения оставим вариант Авто.

Свойство Подсказка содержит текст всплывающей подсказки при кратковременной задержке курсора над командой. Для подсказки зададим значение «Получение печатных форм».

Свойство Картинка содержит картинку, которая будет представлять группу в интерфейсе. Для картинки зададим (аналогично произвольной команде) значение Печать.

В результате наших настроек представление группы команд изменится следующим образом (рис. 1.173) – она представлена назначенной картинкой и текстом, заданным в свойстве Синоним.

Из особенностей объекта Группа команд необходимо отметить, что для него не назначаются права, не определяется принадлежность к подсистемам и зависимость от функциональных опций. Произвольная группа будет представлена в командном интерфейсе пользователя только в том случае, когда в нем доступна хотя бы одна из команд, включенных в группу. В противном случае группа в командный интерфейс не включается.

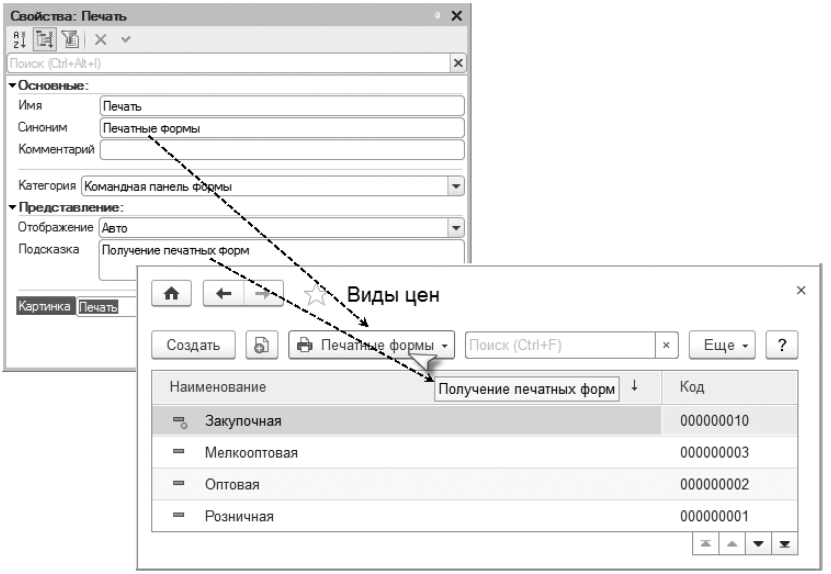


Рис. 1.173. Настройка представления произвольной группы команд

В нашем приложении доступ к справочнику Виды цен разрешен и для роли Менеджер по продажам. Но для этой роли сброшено право Просмотр для произвольной подчиненной команды (рис. 1.174).

В результате такой настройки прав произвольная группа команд Печать для менеджера по продажам оказалась пустой, она не представлена в командной панели формы списка справочника Виды цен.

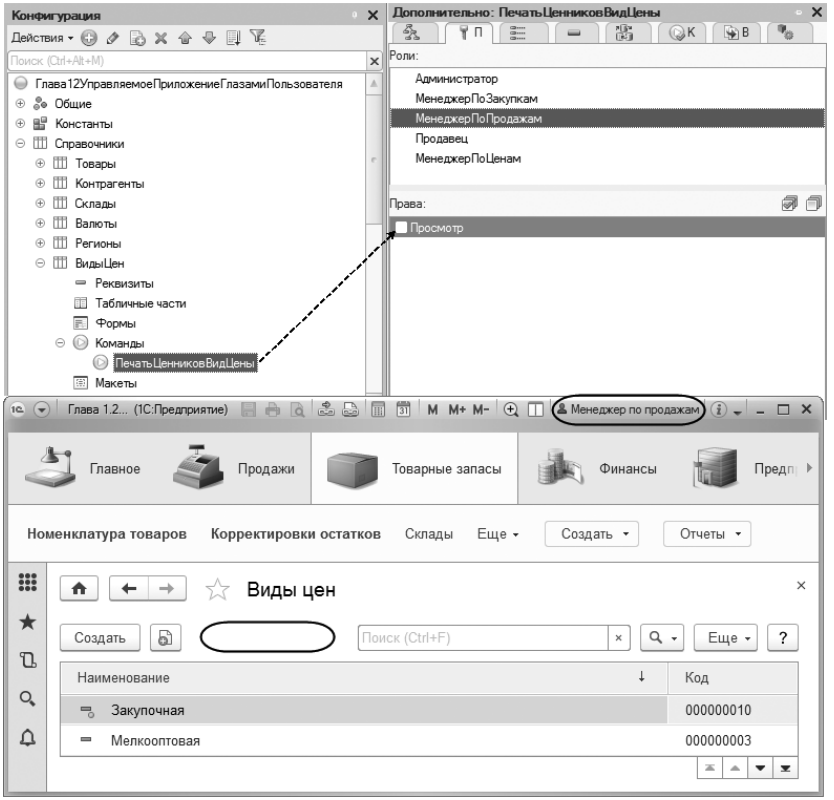


Рис. 1.174. Ролям со сброшенным свойством «Просмотр» произвольная команда недоступна

Глава 1.12. «Командуем» формами

В предыдущих главах мы познакомились с глобальным командным интерфейсом «1С:Предприятия», принципами его построения и глобальными командами.

В данной главе мы рассмотрим формирование набора команд, доступных для использования в контексте формы. Здесь же изложены основные сведения о форме, необходимые для понимания этого механизма.

Необходимые сведения о формах

Формы предназначены для интерактивной работы пользователя с данными информационной базы. Для обеспечения этой возможности форма «наполняется» необходимой функциональностью.

Функциональность формы определяется составом ее реквизитов и команд. *Реквизиты формы* – это данные, с которыми работает форма. *Команды формы* – это действия, которые может выполнить форма над данными.

Однако наличие команды в форме еще не дает возможности использовать ее функциональность. Аналогично и реквизиты сами по себе не обеспечивают возможности отображения и редактирования данных.

Для использования команд для отображения и редактирования данных, хранимых в реквизитах, служат *элементы формы*, связанные с соответствующими командами и реквизитами.

Не вдаваясь в детали, связь между командами, реквизитами и элементами формы можно представить схемой (рис. 1.175).

Можно отметить следующие ключевые особенности форм.

Более подробно о работе с формами «1С:Предприятия» рассказывается во второй части книги – «Конструирование форм» на стр. 235.

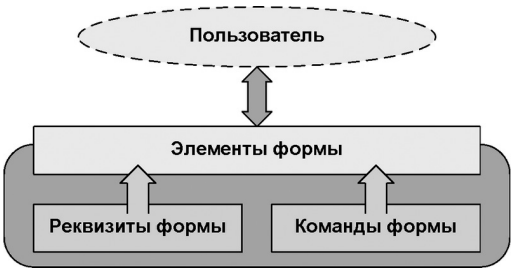


Рис. 1.175. Связь между командами, реквизитами и элементами формы

Во-первых, форма не прорисовывается детально разработчиком, а *строится системой автоматически*. Разработчик же в режиме конфигурирования:

- определяет состав формы в виде дерева элементов;
- описывает поведение формы, задавая значения для ее свойств и/или реализуя процедуры на встроенном языке.

При построении интерфейса для того или иного пользователя система использует это *декларативное описание* для создания формы и размещения ее элементов.

Во-вторых, при создании формы используется *модель управления доступностью и видимостью элементов формы*. При этом учитываются:

- настройки прав в разрезе ролей пользователей;
- зависимость элементов формы от функциональных опций;
- настройка формы, выполненная разработчиком на этапе конфигурирования прикладного решения;
- настройка формы, выполненная пользователем на этапе эксплуатации прикладного решения.

Функциональность по умолчанию

В «1С:Предприятии» можно не создавать формы для представления и обработки объектов данных. В этом случае при выполнении команд открытия форм система самостоятельно, на лету, создаст необходимую форму. Созданная форма будет обладать функциональностью

и представлением по умолчанию. Что же определяет представление и функциональность формы?

Стандартное представление и функциональность формы определяют объект встроенного языка `УправляемаяФорма` *(например, способность формы закрываться) и расширение формы (например, способность записывать данные формы в информационную базу).*

Расширение формы – это дополнительные свойства, методы, параметры и команды, которые появляются у объекта `УправляемаяФорма`, когда ей назначен основной реквизит.

Важно понимать следующее:

- Дополнительно предоставляемая функциональность включается в объект `УправляемаяФорма`, то есть становится ее неотъемлемой частью.
- Состав дополнительных возможностей определяется типом основного реквизита формы, то есть типом тех данных, для редактирования которых предназначена форма.

Расширения могут присутствовать и у элементов формы. Как и для самой формы, состав расширения, то есть дополнительные свойства, методы и т. д., элемента формы определяется типом реквизита, с которым элемент связан.

При необходимости реализовать нестандартное представление данных или нестандартную функциональность разработчик может самостоятельно разработать форму в конфигураторе. Для разработки форм используется *редактор форм*. С помощью этого редактора разработчик создает для формы необходимый набор реквизитов и команд, а также элементы формы, которыми они отображаются.

Более подробно о работе с редактором форм можно прочитать в главе 2.3 на стр. 256.

В качестве отправной точки мы будем использовать демонстрационную базу «Глава 1.11. Создаем произвольные команды». Результат выполняемых действий можно посмотреть в демонстрационной базе «Глава 1.12. «Командуем» формами».

Команды формы

Для знакомства с командами формы создадим еще одну форму для документа Расход товара. Из контекстного меню узла Формы этого документа выберем пункт Добавить.

В результате откроется окно конструктора форм (рис. 1.176). В окне конструктора выберем тип формы – Форма документа, установим флажок Назначить форму основной и зададим имя ОсновнаяФорма. Нажмем кнопку Готово.

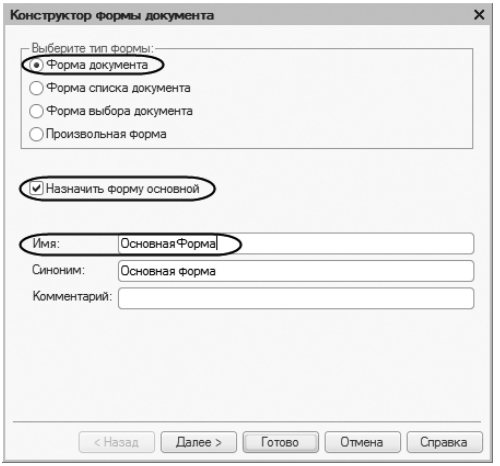


Рис. 1.176. Окно конструктора формы

В результате работы конструктора будет создана форма документа, эта форма назначена основной, то есть она используется по умолчанию для работы с документом. После завершения работы конструктора для созданной формы откроется окно редактора формы (рис. 1.177).

В редакторе формы основной реквизит формы выделяется жирным шрифтом.

Если открыть документ РасходТовара (Продажа) в режиме 1С:Предприятие, мы увидим, что для работы с документом используется созданная нами форма (рис. 1.178).

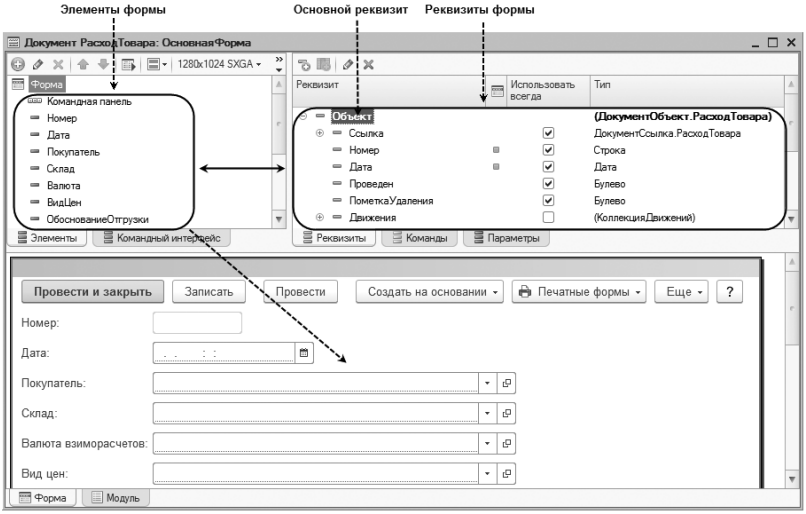


Рис. 1.177. Редактор формы с автоматически созданной формой документа

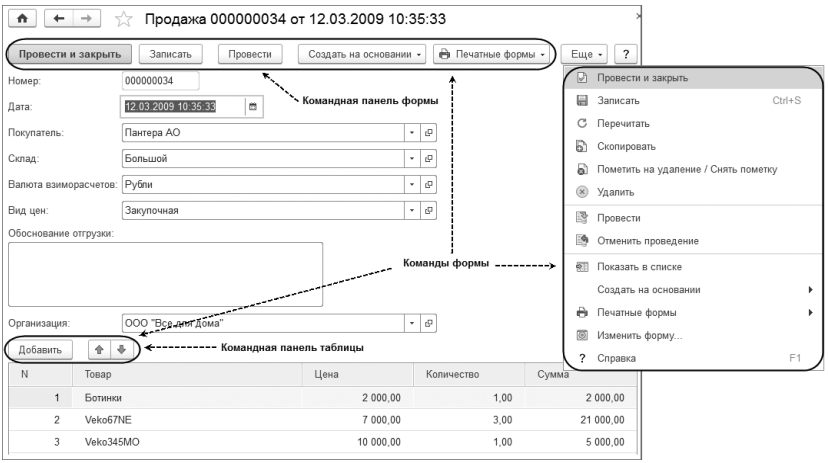


Рис. 1.178. Форма редактирования документа «Расход товара»

Элементы, обеспечивающие доступ к командам, расположены в командных панелях. В нашем случае система сформировала командную панель формы и командную панель таблицы товаров. Выбор любой из доступных команд можно осуществить из подменю

Еще соответствующей командной панели. Для ускорения доступа к командам часть из них (наиболее важные или часто используемые) представлена кнопками непосредственно в командных панелях.

Чем «руководствуется» система при формировании состава команд формы? Какие команды должны быть в форме? Для ответа на эти вопросы нужно вспомнить основное назначение формы – интерактивная обработка данных. Следовательно, в форме должны присутствовать команды, предоставляющие пользователю возможность обработать данные формы и возможность обратиться к данным, связанным с обрабатываемыми.

Для обработки данных формы – стандартные команды формы

В форме должны присутствовать команды для обработки данных и для управления формой. Эти возможности обеспечивают *стандартные локальные команды формы и ее элементов*. В редакторе форм они представлены на закладке Стандартные команды редактора команд (рис. 1.179).

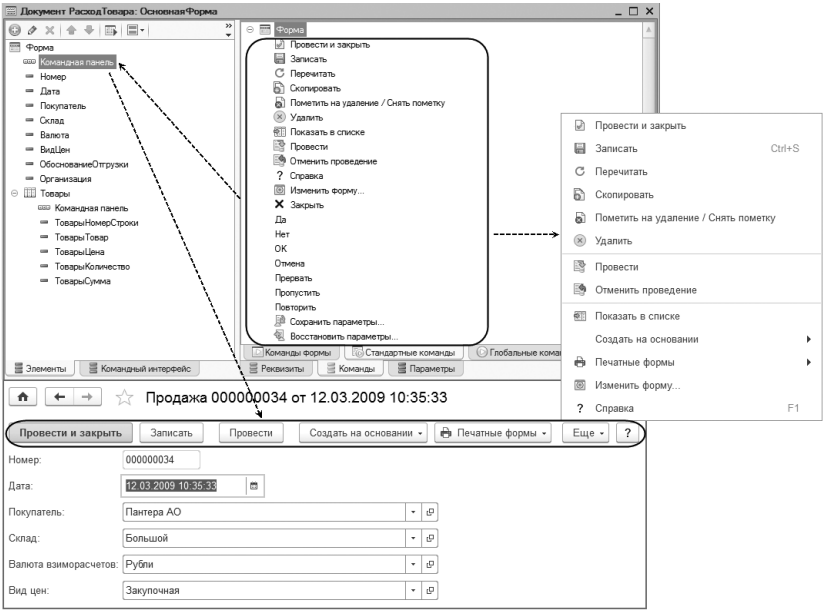


Рис. 1.179. Стандартные команды формы в редакторе и интерфейсе

Эти команды *предоставлены формой и расширением формы, а также элементами, расположенными в форме, которые являются источниками команд.* Например, таблицами и полями: форматированного документа, табличного документа, графической схемы, планировщика.

Состав команд, предоставляемых формой, является стандартным и не зависит от данных формы – это команды Справка, Показать в списке, Изменить форму..., Закрыть, Сохранить параметры..., Восстановить параметры...

Состав команд, предоставляемых расширением формы, зависит от типа основного реквизита формы. В нашем случае основным реквизитом формы назначен реквизит Объект с типом данных ДокументОбъект.РасходТовара (см. рис. 1.177). Расширение, соответствующее этому типу данных, предоставило команды Провести и закрыть, Записать, Перечитать, Скопировать, Пометить на удаление / Снять пометку, Удалить, Провести и Отменить проведение.

Если в составе элементов формы присутствуют таблицы, форматированные документы, табличные документы и др. источники команд, то в состав стандартных локальных команд формы добавляются команды, обеспечивающие обработку данных, отображаемых в этих элементах.

У документа РасходТовара есть табличная часть, которая в данных формы представлена реквизитом Товары. Для отображения списка товаров в форме используется элемент Товары, имеющий тип Таблица. В состав стандартных локальных команд формы включены команды обработки табличных данных – узел Товары в редакторе команд (рис. 1.180).

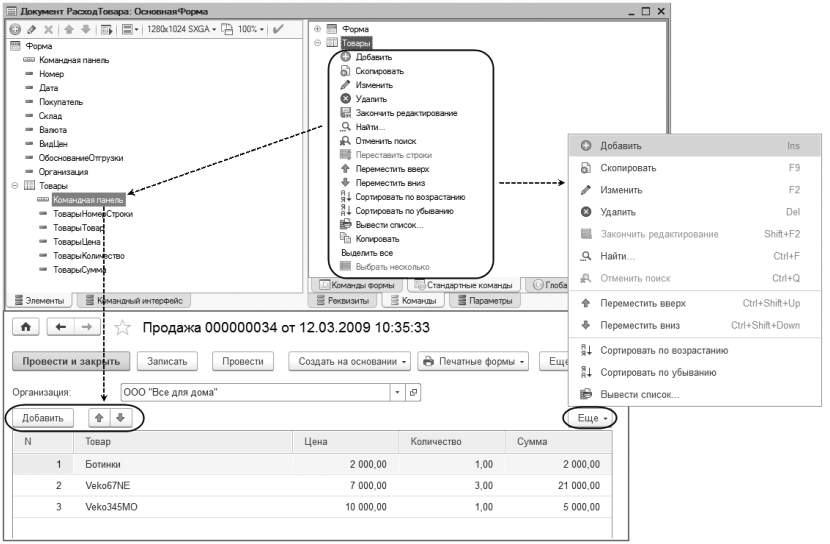


Рис. 1.180. Стандартные команды таблицы в редакторе и интерфейсе

Для работы со связанными данными – глобальные параметризуемые команды

При обработке данных формы может возникнуть необходимость просмотреть данные, связанные с обрабатываемыми. Это может быть, например, набор записей в регистре взаиморасчетов с контрагентами, подчиненный обрабатываемому документу, или список цен на продаваемый товар и др.

Также может потребоваться выполнить какую-либо обработку связанных данных – например, ввести документ оплаты на основании документа продажи или напечатать штрихкоды продаваемого товара и др.

Доступ к связанным данным обеспечивают *глобальные параметризуемые навигационные команды*, а обработку связанных данных – *глобальные параметризуемые команды действий*. В редакторе форм они представлены на закладке Глобальные команды редактора команд (рис. 1.181).

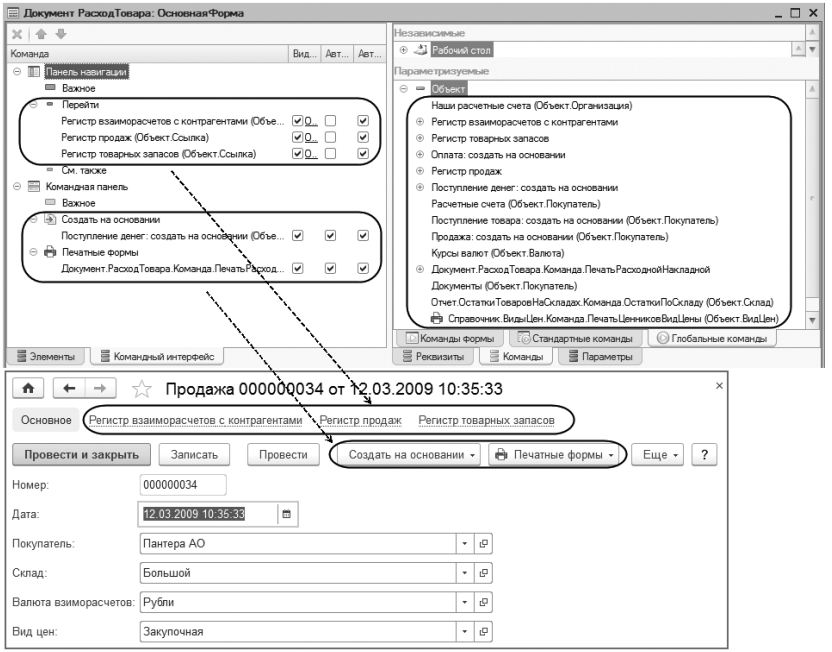


Рис. 1.181. Глобальные параметризуемые команды в редакторе и интерфейсе

Эти команды предоставлены глобальным командным интерфейсом прикладного решения. Состав команд, доступный в форме, зависит от типа параметра параметризованной команды (см. раздел «Произвольные команды» на стр. 149). В форме доступны те глобальные параметризованные команды, для которых в контексте формы можно получить значения параметра требуемого типа.

В редакторе команд источник параметра для команды указывается в скобках после команды. Например, для команды открытия списка регистра продажи параметром выступает ссылка на редактируемый документ.

Если для команды в форме существует несколько источников, то команда представлена как узел дерева, а список источников – как элементы этого узла. В состав команд формы такую команду можно включить с любым из источников (или несколько экземпляров одной команды с разными источниками).

Для работы с функциональностью приложения – глобальные независимые команды

При обработке данных формы может возникнуть необходимость воспользоваться функциональностью приложения, не связанной напрямую с обрабатываемыми данными.

Например, при обработке данных документа нам необходимо выполнить поиск в данных или ввести новый вид цен. Выполнить эти операции помогут *глобальные независимые команды*. В редакторе форм они представлены на закладке Глобальные команды редактора команд (рис. 1.182).

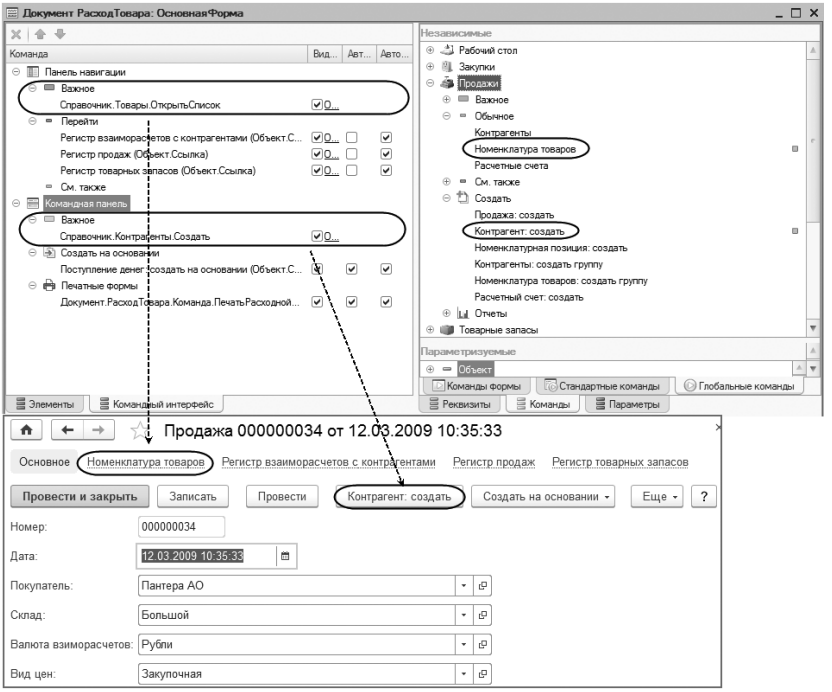


Рис. 1.182. Глобальные независимые команды в редакторе и интерфейсе

Глобальные независимые команды предоставлены глобальным командным интерфейсом. В редакторе команд глобальные независимые команды сгруппированы по разделам глобального командного интерфейса. Команда будет доступна из всех разделов глобального командного интерфейса, в которые она включена.

Способы формирования состава команд формы

Познакомившись с источниками команд формы, давайте посмотрим, какие варианты предоставляет нам система для формирования состава команд формы.

В самом общем случае существует три варианта:

- Автоматический – состав команд формы определяется системой полностью автоматически.
- Комбинированный – состав команд формы определяется системой автоматически, а разработчик, используя редактор форм, корректирует его.
- Ручной – состав команд формы определяется полностью разработчиком.

Для формы есть еще один «источник команд» – разработчик, который может создавать произвольные локальные команды формы. Об этих командах мы поговорим немного позже (см. раздел «Если не хватает стандартных команд» на стр. 226).

Автоматический вариант является самым быстрым и наименее затратным с точки зрения разработки прикладного решения: система все делает самостоятельно.

Комбинированный вариант, вероятно, используется чаще всего. При таком подходе разработчик вмешивается в работу системы только тогда, когда набор команд, сформированный системой, не обеспечивает требуемых функциональных возможностей по обработке данных формы. В подавляющем большинстве случаев вмешательство выражается в простом расширении состава команд формы.

Ручной же вариант предоставляет максимум возможностей по управлению составом команд формы и их размещением. Однако он требует от разработчика выполнения значительного объема кропотливой работы.

Автоматическое формирование состава команд формы

При автоматическом формировании состава команд система включает в форму:

- стандартные команды формы;
- глобальные параметризуемые команды, для параметров которых в форме есть значения подходящего типа.

ВНИМАНИЕ!

Глобальные независимые команды автоматически в форму не включаются.

Для команд из различных источников система использует разные правила видимости по умолчанию:

- все стандартные команды формы видимы;
- все глобальные параметризуемые команды действия видимы;
- все глобальные параметризуемые навигационные команды невидимы.

Для автоматического размещения команд система использует панель навигации окна клиентского приложения и командную панель формы (рис. 1.183).

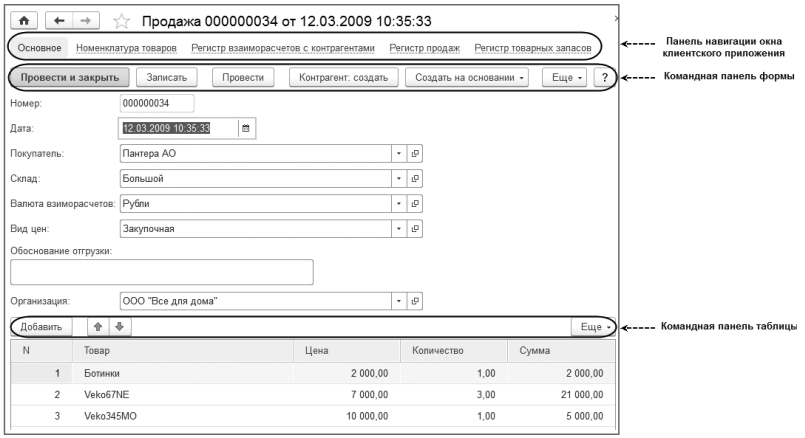


Рис. 1.183. Форма документа «РасходТовара (Продажа)»

Стандартные команды формы

Стандартные команды формы автоматически размещаются в командной панели формы. В нее всегда включаются все команды, предоставленные расширением формы и ее элементов, а также команды, предоставленные самой формой (рис. 1.184).

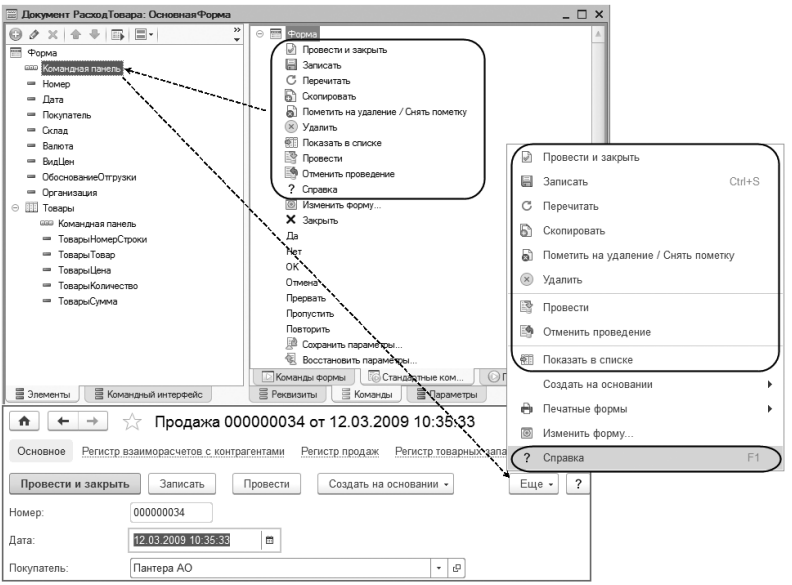


Рис. 1.184. Командная панель формы заполнена автоматически

Включение в командную панель команды Изменить форму... определяется значением свойства формы Разрешить изменять форму. По умолчанию свойство имеет значение Истина и команда включается в командную панель (рис. 1.185).

Включение в командную панель команд Сохранить параметры... и Восстановить параметры... определяется значением свойства формы Сохранение данных в настройках. По умолчанию это свойство имеет значение Не использовать и команды не включаются в командную панель (см. рис. 1.185).

Часть команд отображается непосредственно в командной панели и в подменю Еще, а часть – только в подменю Еще.

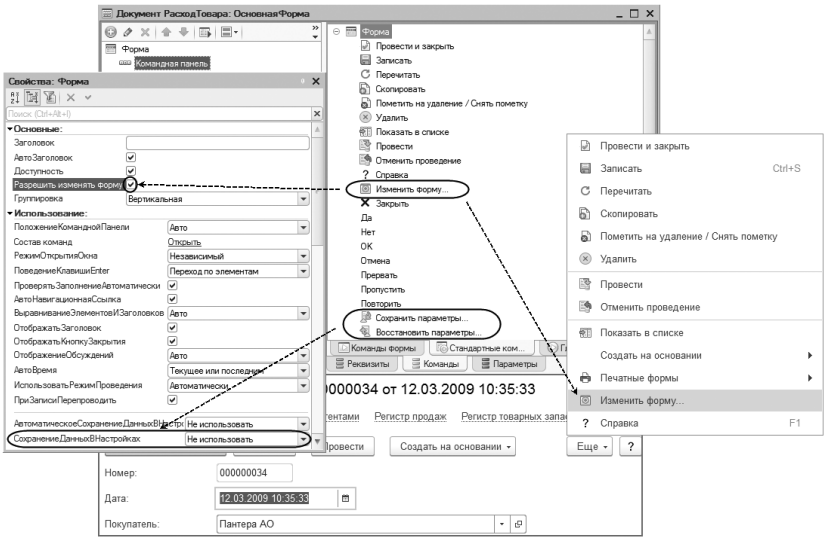


Рис. 1.185. Команды настройки формы

Если в составе элементов формы присутствуют таблицы (а также форматированные документы, табличные документы и др. источники команд), то для размещения команд обработки табличных данных также используется командная панель. В дереве элементов формы она располагается сразу под соответствующим табличным элементом (рис. 1.186).

При автоматическом конструировании формы списка командная панель таблицы, отображающей данные основного реквизита формы, отключается (свойство Автозаполнение=Ложь), а все ее команды платформа помещает в командную панель формы.

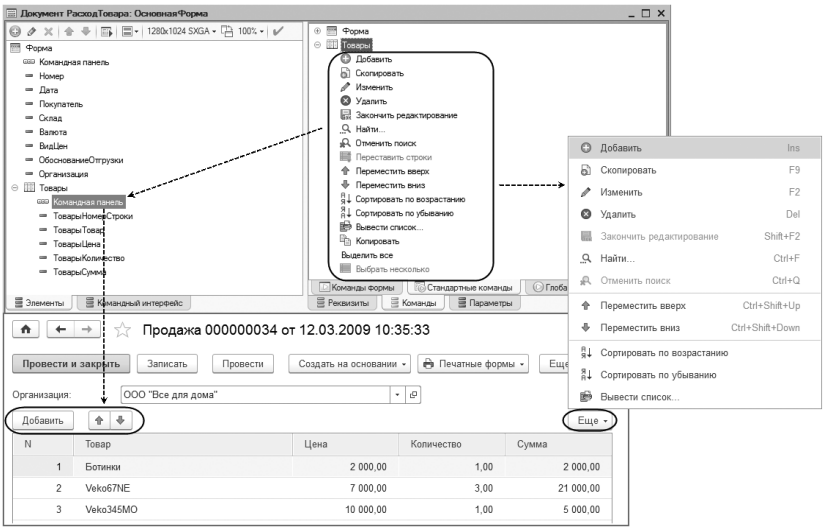


Рис. 1.186. Команды обработки табличных данных

Глобальные параметризуемые команды

При автоматическом формировании состава команд в форму включаются только те глобальные параметризуемые команды, которые в качестве параметра могут принимать ссылку на основной реквизит формы или на реквизиты основного реквизита формы.

Размещение по умолчанию глобальных параметризуемых команд определяется их категорией и группой.

Категории стандартных команд предопределены и не могут быть изменены. А вот группу (внутри категории), в которой будет размещена команда, разработчик может изменить.

Категория и группа произвольных (как общих, так и подчиненных) команд определяются разработчиком путем выбора значения в свойстве Группа этой команды (см. раздел «Особенности размещения» на стр. 155).

Разработчик может настроить видимость автоматически включенных в форму глобальных параметризуемых команд. А вот удалить их система не позволит.

Глобальные *параметризуемые команды действий* размещаются в командной панели формы. В редакторе формы эти команды отображаются на закладке Командный интерфейс редактора команд (рис. 1.187).

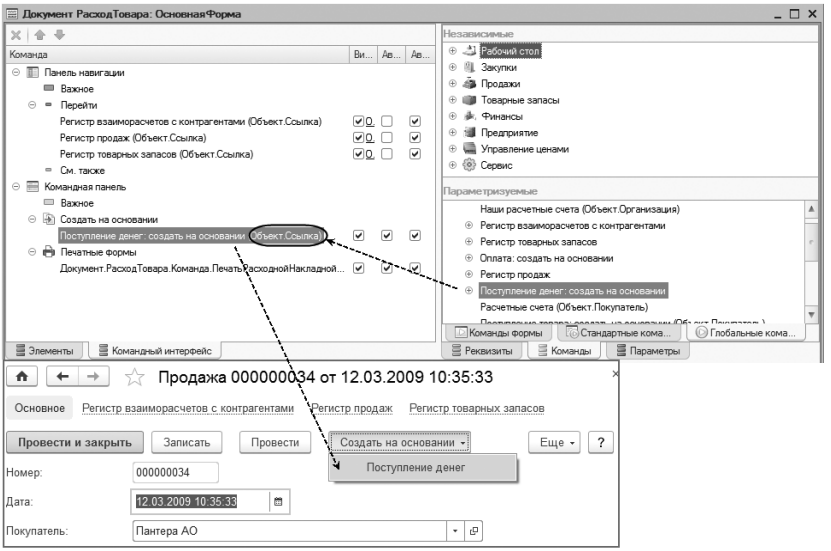


Рис. 1.187. Глобальная параметризуемая команда действий в форме

Для включения произвольной глобальной параметризуемой команды в командную панель формы в свойстве Группа этой команды должна быть выбрана группа команд с категорией Командная панель формы.

Например, в созданной нами форме в командную панель формы включена команда Поступление денег, которая создает соответствующий документ на основании обрабатываемого документа расхода (см. рис. 1.187). Для команды в качестве параметра передается ссылка на обрабатываемый документ (из свойства Ссылка основного реквизита формы).

Глобальные *параметризуемые навигационные команды* размещаются в панели навигации окна клиентского приложения. В редакторе формы эти команды отображаются на закладке Командный интерфейс редактора команд (рис. 1.188).

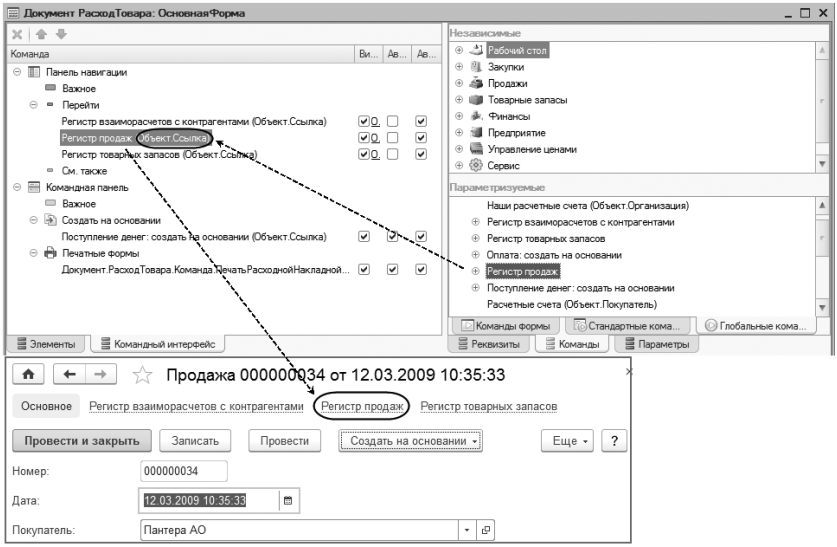


Рис. 1.188. Глобальная параметризуемая навигационная команда

Для включения произвольной глобальной параметризуемой команды в панель навигации окна клиентского приложения в свойстве Группа этой команды должна быть выбрана группа команд с категорией Панель навигации формы.

По умолчанию эти команды невидимы и панель навигации не отображается. Отображение этих команд в панели навигации окна клиентского приложения мы настроили в конфигураторе.

Например, в созданной нами форме в панель навигации окна клиентского приложения включена команда Регистр продаж, которая открывает список записей регистра продаж, сформированных обрабатываемым документом (см. рис. 1.188). Для команды в качестве параметра передается ссылка на обрабатываемый документ (из свойства Ссылка основного реквизита формы).

Доступность команд формы для пользователя

При создании формы для пользователя прикладного решения система учитывает его права, определяемые ролью, и значения функциональных опций. Из этого следует, что набор команд формы, доступный конкретному пользователю, может отличаться от состава команд, включенных в форму на этапе разработки.

Как и в случае с видимостью, для команд из разных источников система использует разные правила для автоматического определения их доступности:

- Доступность стандартных команд, предоставленных формой, не зависит от ролевой настройки прав и значений функциональных опций.
- Доступность стандартных команд, предоставленных расширением основного реквизита формы, зависит от ролевой настройки прав пользователя, а значения функциональных опций на доступность команд не влияют.
- Доступность глобальных команд зависит от ролевой настройки прав пользователя и значений функциональных опций.

Для демонстрации влияния ролевой настройки прав на доступный пользователю набор команд включим документ РасходТовара в подсистему Ценообразование (механизм включения описан в разделе «Стандартные команды» на стр. 56).

Для роли Менеджер по ценам установим права Чтение и Просмотр на документ. Для подчиненной документу произвольной команды Печать расходной накладной право Просмотр устанавливать не будем (механизм установки прав описан в разделе «Система прав доступа» на стр. 65).

Чтобы исключить ошибки времени выполнения, менеджеру по ценам установим право Чтение на справочники Склады, Организации и Валюты.

Для пользователя Администратор установлены все права на документ Расход товара. При построении формы документа (и формы списка документов) в состав доступных включены все стандартные команды формы, включенные в нее на этапе разработки (рис. 1.189 вверху).

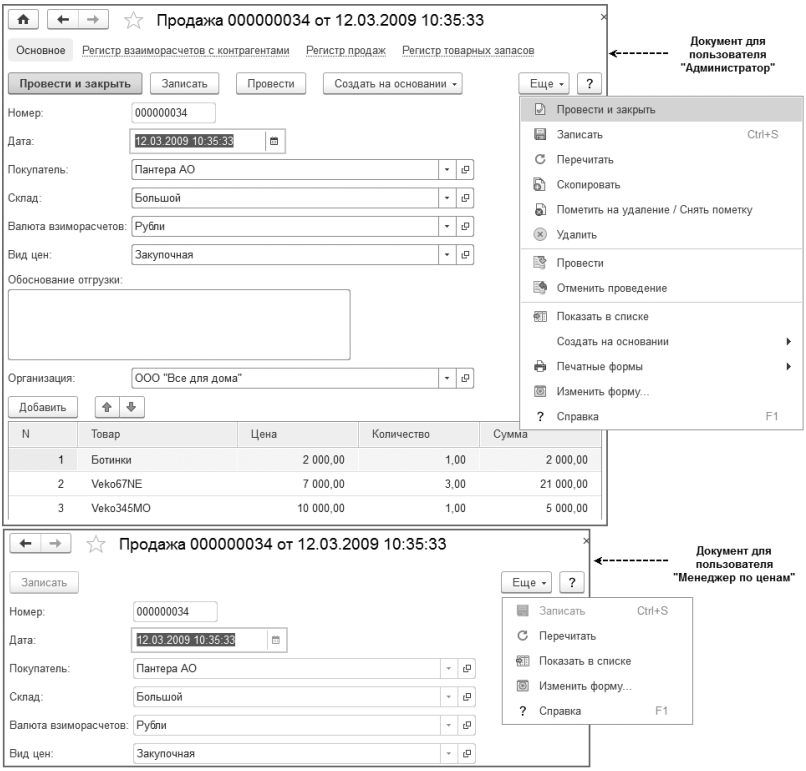


Рис. 1.189. Стандартные команды, доступные пользователям с разными ролями

Для пользователя же с ролью Менеджер по ценам разрешен только просмотр документов Расход товара. При построении формы документа (и формы списка документов) для этого пользователя система исключила из состава доступных команды расширения основного реквизита, выполняющие запрещенные пользователю операции с документом: создания, удаления, проведения документа и т. д. (см. рис. 1.189 внизу).

Доступность глобальных команд в форме определяется их доступностью в глобальном командном интерфейсе, который формируется при запуске системы от имени того или иного пользователя. Это значит, что команды, недоступные пользователю в глобальном командном интерфейсе, также недоступны ему в форме.

Например, для пользователя Администратор установлено право на использование команды печати документа РасходТовара. При построении формы документа (и формы списка документов) в состав доступных включена эта глобальная команда (рис. 1.190 вверху).

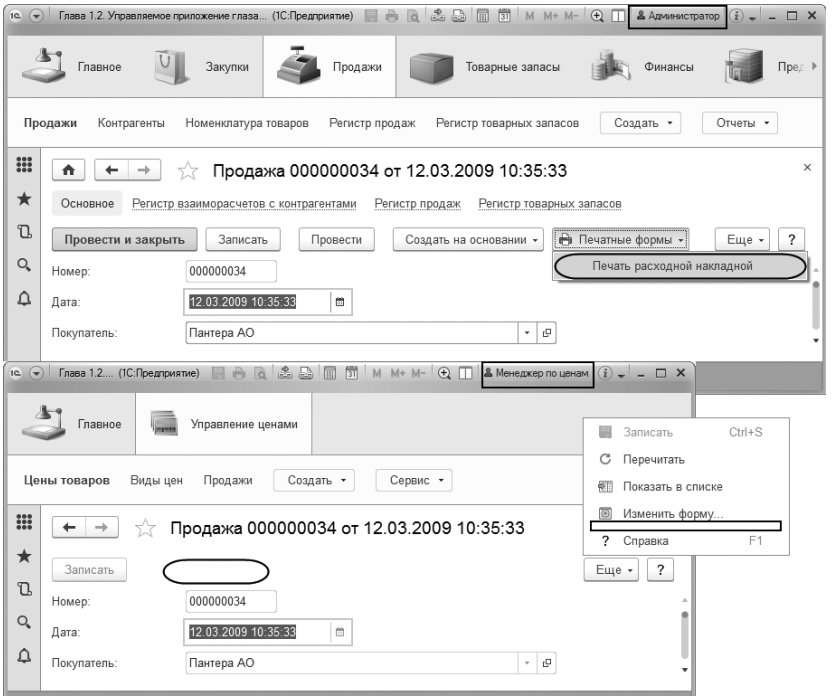


Рис. 1.190. Глобальные команды, доступные пользователям с разными ролями

В отличие от администратора пользователь Менеджер по ценам не имеет права на печать документа РасходТовара. При построении формы документа (и формы списка документов) для этого пользователя система исключила из состава доступных команду печати расходной накладной (см. рис. 1.190 внизу).

Помимо набора прав на состав глобальных команд, доступных пользователю в форме, влияют значения функциональных опций. Для демонстрации их влияния на доступность команд формы

вспомним, что ранее мы добавили (см. раздел «Произвольные группы» на стр. 180) произвольную команду Печать ценников для вида цены, подчиненную справочнику Виды цен, в группу Печатные формы командной панели.

Для этой команды установлена зависимость от функциональной опции Ценообразование по видам цен.

В случае, когда эта функциональная опция имеет значение Истина, команда печати ценников для вида цен включена в состав доступных команд формы (рис. 1.191).

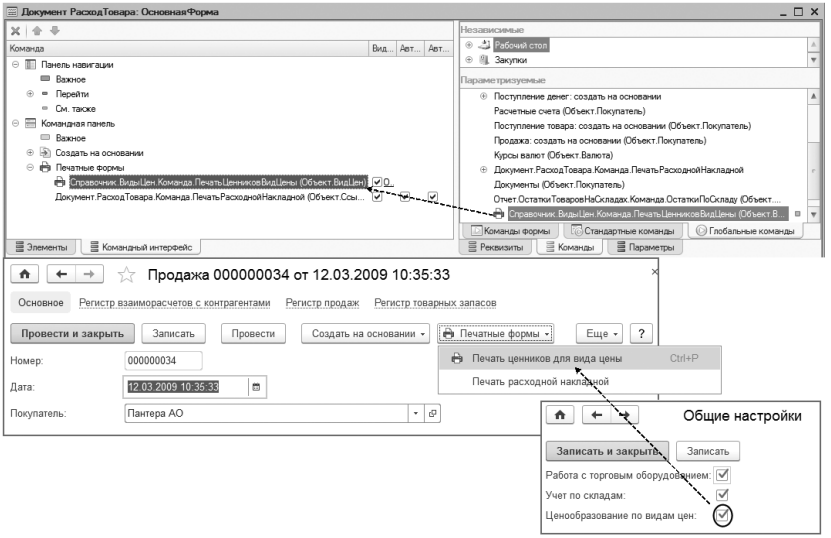


Рис. 1.191. При истинном значении функциональной опции команда печати ценников доступна

В случае же, когда функциональная опция имеет значение Ложь, команда в форме недоступна, хотя в редакторе она включена в состав команд формы (рис. 1.192).

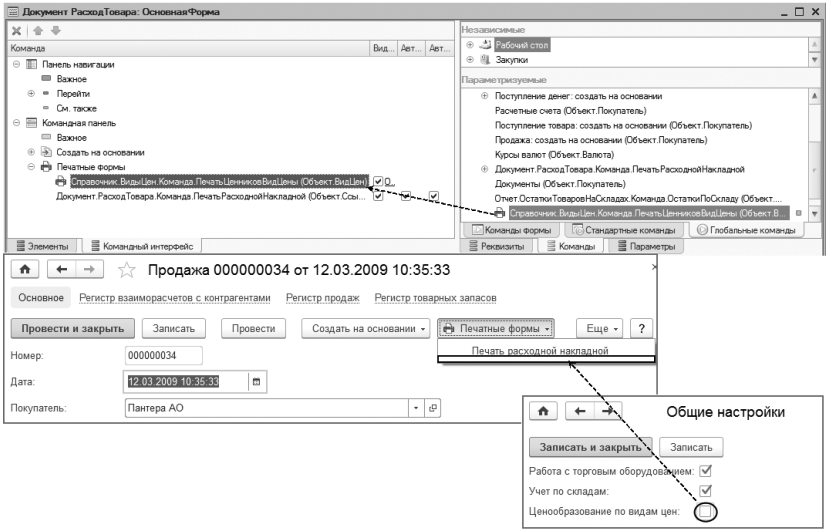


Рис. 1.192. При ложном значении функциональной опции команда печати ценников недоступна

Управляем командами формы

Начнем с настройки командной панели формы. Ее отображением для пользователя (и ее положением) может управлять разработчик. Вариант размещения командной панели в форме определяется значением свойства формы Положение командной панели. По умолчанию значением свойства является Авто и командная панель присутствует в форме. Если же для свойства установить значение Нет, то в форме командной панели не будет (рис. 1.193).

Естественно, в этом случае будут недоступны все команды, размещаемые в командной панели формы.

Кроме возможности полного отключения командной панели разработчик может управлять вариантом ее «наполнения» командами. Для этого используется свойство командной панели Автозаполнение.

В случае, когда флажок Автозаполнение сброшен (рис. 1.194), состав команд панели формируется разработчиком вручную. Если отображение командной панели включено, но командная панель пустая, то в форме она не отображается.

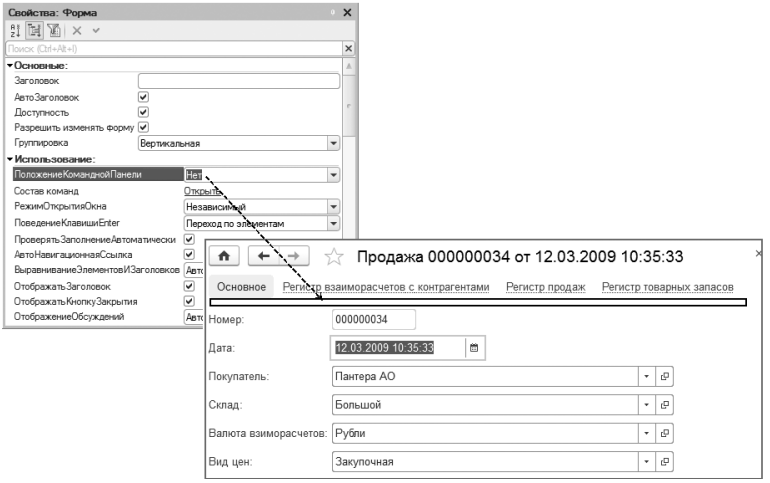


Рис. 1.193. Управление наличием командной панели в форме

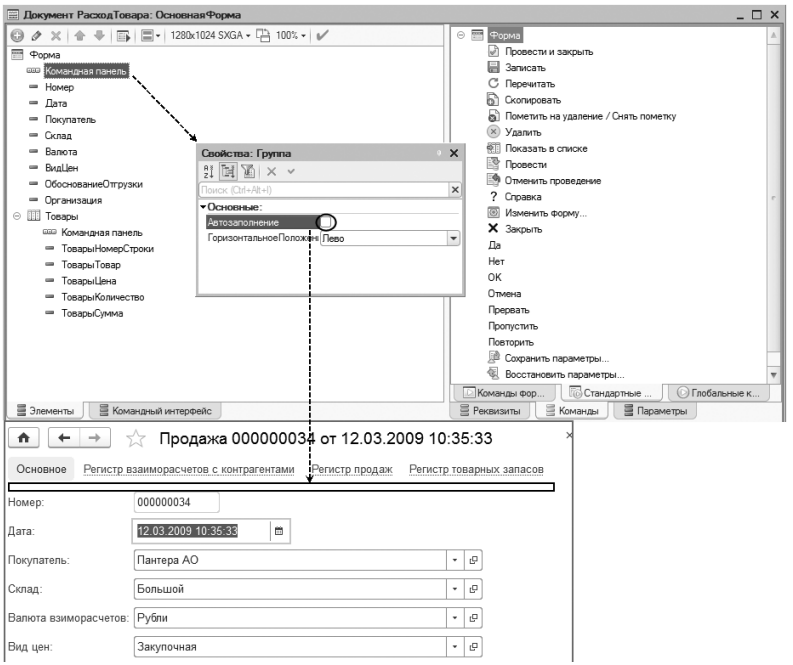


Рис. 1.194. Автозаполнение у командной панели формы отключено

При установленном флажке состав команд формируется автоматически – так, как было описано ранее. Но и в случае автозаполнения система позволяет разработчику повлиять на состав команд, включаемых в командную панель формы (и в командную панель таблицы).

Мы не будем формировать набор команд полностью в ручном режиме. Нам достаточно разобраться в том, как этот набор изменить (по сравнению с автоматическим режимом).

Комбинируя возможность автозаполнения командной панели и возможность ручного управления составом команд, можно с минимальными затратами времени и сил сформировать требуемый состав команд формы. Включим снова автозаполнение командной панели формы (если оно было выключено).

Стандартные команды формы

Состав используемых стандартных команд формы можно определить в свойстве формы (и в свойстве таблицы) Состав команд. Настройка выполняется в окне Состав, вызываемом через гиперссылку Открыть (рис. 1.195).

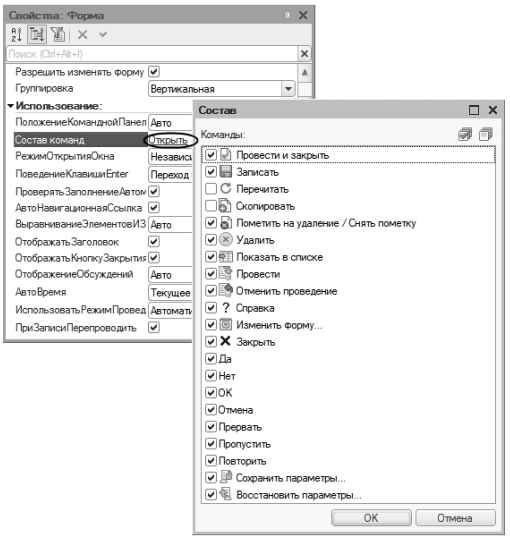


Рис. 1.195. Настройка состава доступных стандартных команд формы

Для исключения команды необходимо снять флажок рядом с соответствующей командой. Снимем флажки у команд Перечитать и Скопировать. Эти команды исключены и из состава доступных пользователю, и из состава команд формы в редакторе команд, то есть они становятся недоступными и разработчику (рис. 1.196).

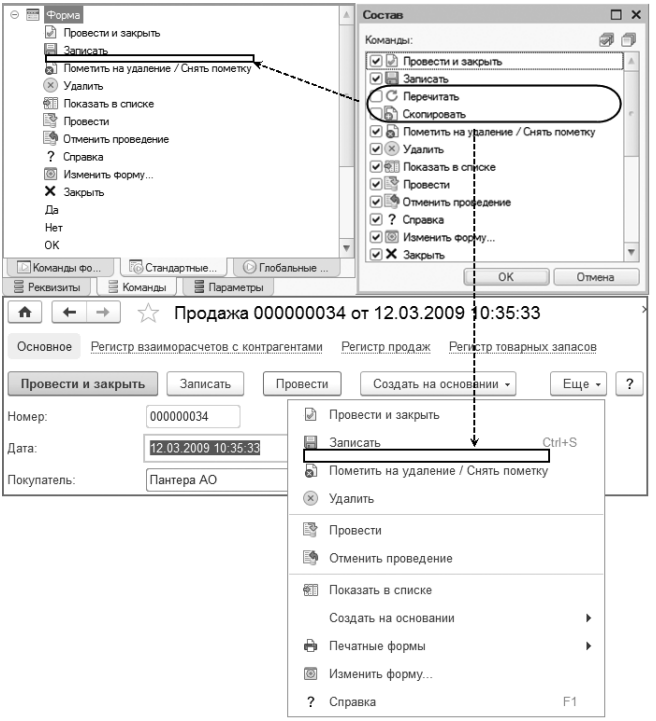


Рис. 1.196. «Выключенные» стандартные команды в интерфейсе отсутствуют

Теперь отобразим в форме стандартную команду (например, команду Закрыть), которая ранее не отображалась.

Для добавления команды в форму необходимо создать элемент формы Кнопка, который позволяет выполнить команду, и связать с этим элементом добавляемую команду.

При формировании состава командной панели свойство Состав команд имеет более высокий приоритет, чем свойства Разрешить изменять форму и Сохранение данных в настройках.

Обычно кнопка располагается в командной панели – подчинена узлу Командная панель в дереве элементов формы. Команда, вызываемая кнопкой, задается в свойстве кнопки ИмяКоманды.

Проще всего перетащить команду из окна редактора команд в командную панель или в дерево элементов формы. В этом случае связывание кнопки и команды выполняется автоматически. Перетащим команду Закрыть в командную панель формы.

В результате в дереве элементов появился новый элемент формы с именем ФормаЗакрыть вида Кнопка командной панели, подчиненный элементу Командная панель, и открылось окно свойств добавленного элемента. Свойство ИмяКоманды автоматически заполнено значением Закрыть – командой, которую мы перетащили в командную панель (рис. 1.197).

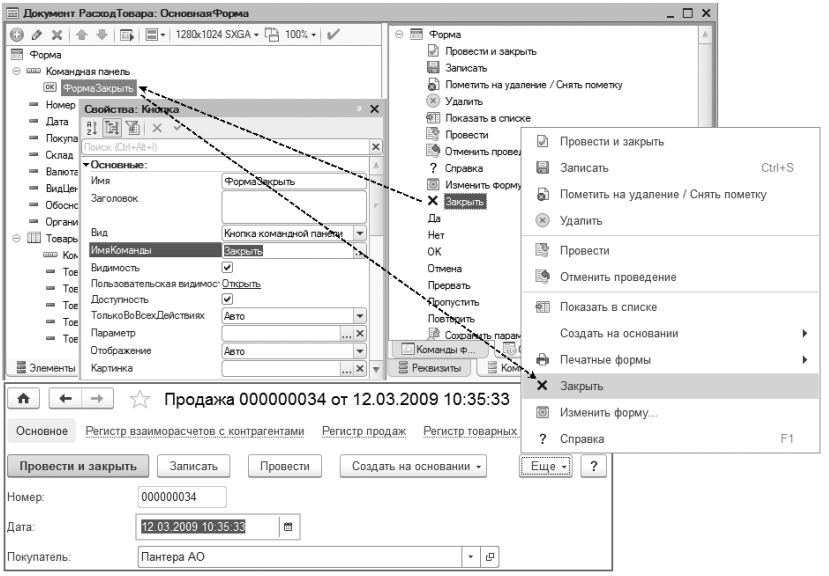


Рис. 1.197. Форма с добавленной вручную командой «Закрыть»

Для добавленной вручную команды разработчик может настроить доступность и видимость, ее расположение и представление, вариант отображения команды и другие аспекты ее поведения.

Настройка выполняется путем задания значений свойствам того элемента формы, с которым команда связана.

ВНИМАНИЕ!

Фактически выполняется настройка не самой команды, а элемента формы, с которым команда связана.

Место отображения добавленной команды в командной панели можно настроить. Для этого используется свойство кнопки Только во всех действиях:

- Да – команда размещается только в подменю Еще;
- Нет – команда размещается и в подменю Еще, и в командной панели;
- Авто – решение о месте размещения команды принимается системой.

Для кнопки ФормаЗакрыть оставим значение по умолчанию Авто (см. рис. 1.198) – команда будет доступна и непосредственно в командной панели, и в подменю Еще.

ВНИМАНИЕ!

Если все доступные команды отображаются в командной панели, подменю Еще в командной панели отсутствует.

Для настройки представления команды используются свойства Заголовок, Отображение, Вид, Фигура, Положение картинки и др. свойства кнопки.

Свойство Заголовок позволяет задать текст, представляющий команду. Для кнопки ФормаЗакрыть зададим текст «Закрыть форму».

Свойство Вид позволяет задать вариант отображения кнопки в виде обычной кнопки или гиперссылки. Установим свойство Вид у кнопки ФормаЗакрыть как Гиперссылка командной панели.

Свойство Отображение позволяет задать вариант представления команды:

- Текст – в представлении команды присутствует только текст;
- Картинка – в представлении команды присутствует только пиктограмма;
- Картинка и текст – в представлении команды присутствуют и текст, и пиктограмма;
- Авто – решение о представлении команды принимает система.

Для кнопки ФормаЗакрыть зададим значение Текст. В результате наших настроек представление команды Закрыть изменилось (рис. 1.198).

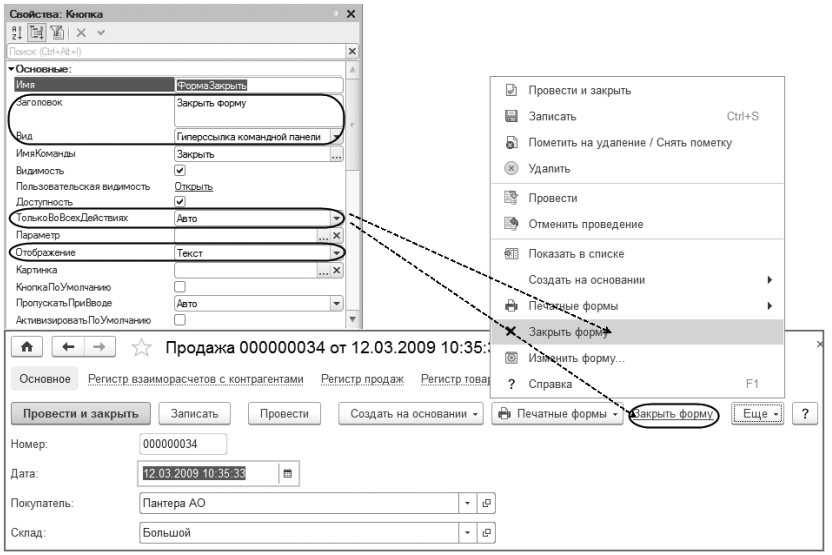


Рис. 1.198. Измененное представление команды «Закрыть»

Команда ФормаЗакрыть отображается в командной панели в виде гиперссылки с текстом «Закрыть форму». В подменю Еще, если для кнопки определена пиктограмма, в любом случае отображаются и текст, и пиктограмма.

Кроме размещения может потребоваться настроить доступность команды (без удаления ее из командной панели). Для этого используется свойство Доступность кнопки, с которой связана команда. Если для команды Закрыть снять флажок в свойстве Доступность, то команда станет недоступной для выбора пользователем. В интерфейсе недоступные команды имеют «бледный» вид – отображаются серым цветом (рис. 1.199).

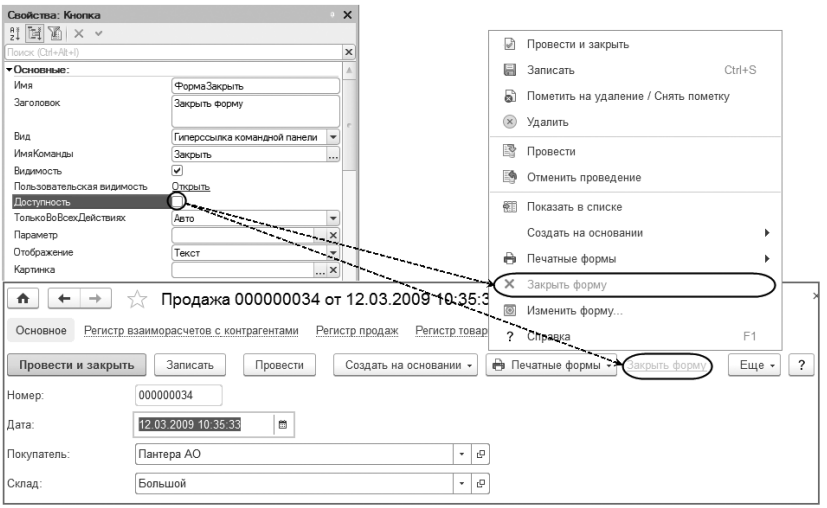


Рис. 1.199. Командная панель с недоступной командой «Закрыть»

Кроме доступности разработчик может управлять и видимостью команды. Для этого используются свойства кнопки Видимость и Пользовательская видимость (рис. 1.200).

Свойство Видимость предназначено для настройки видимости команды в соответствии с реализуемой разработчиком логикой работы прикладного решения. Настройка видимости выполняется установкой (команда видима) или снятием (команда невидима) флажка у свойства.

Свойство Пользовательская видимость определяет, какие команды в форме будут видимы по умолчанию, а какие по умолчанию скрыты. Состав видимых по умолчанию команд можно настроить в разрезе

ролей. Настройка пользовательской видимости выполняется в окне Настройка видимости, вызываемом через гиперссылку Открыть свойства (см. рис. 1.200).

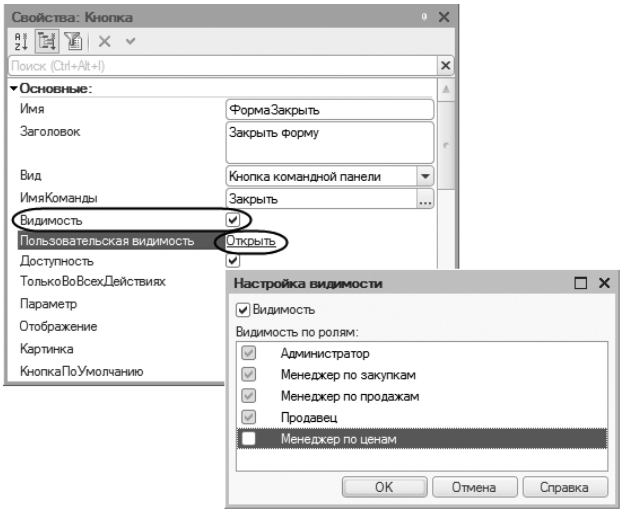


Рис. 1.200. Свойства, управляющие видимостью команды

При настройке видимости можно:

- установить общую видимость по умолчанию – флажок Видимость; определяет видимость для тех ролей, у которых флажок установлен в «третье» значение (серый фон);
- установить видимость по умолчанию команды для конкретной роли пользователя – если рядом с ролью в списке Видимость по ролям флажок установлен, то команда видима для роли; если сброшен, то невидима; если же флажок установлен в «третье» значение (на сером фоне), то для роли используется значение общей видимости.

При определении видимости по умолчанию для конкретного пользователя значения, установленные для его ролей, складываются по логическому ИЛИ.

При эксплуатации прикладного решения пользователь видит только те команды (и др. элементы формы), для которых разработчик установил свойства Видимость и Пользовательская видимость

и которые доступны ему в соответствии с его ролями и не отключены функциональными опциями.

Затем в режиме 1С:Предприятие каждый конкретный пользователь может выполнить настройку пользовательской видимости «под себя» и тем самым скрыть те команды / элементы формы, которыми он не пользуется.

Таким образом, установка свойства Видимость – это инструмент разработчика, а установка свойства Пользовательская видимость – инструмент пользователя. При этом разработчик может задать в конфигураторе пользовательскую видимость в разрезе ролей по умолчанию, которую пользователь может затем донастроить по своему усмотрению.

ПРИМЕЧАНИЕ

Для вызова окна настройки используется команда Изменить форму... Если пользователь перенастроил видимость команд по умолчанию, то система при создании формы для определения видимости команд будет учитывать и настройки разработчика, и настройки пользователя. Причем настройки пользователя будут накладываться на стандартные настройки, сделанные разработчиком.

Давайте для команды Закрыть свойство Видимость установим, а пользовательскую видимость определим следующим образом (см. рис. 1.200):

- Общая видимость установлена.
- Для роли Менеджер по ценам видимость сброшена.
- Для остальных ролей видимость имеет «третье» значение.

Теперь при запуске системы от имени менеджера по закупкам команда Закрыть по умолчанию невидима, а для остальных ролей она видима (рис. 1.201).

При настройке размещения команды в командной панели можно объединять в логические группы. Для этого используется элемент формы с типом Группа.

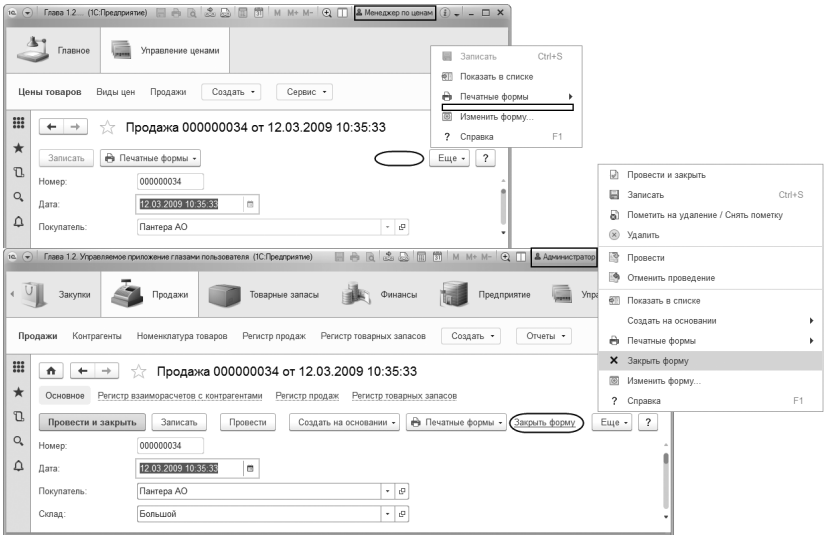


Рис. 1.201. Ограничение видимости команды по умолчанию для различных ролей

Давайте выделим в отдельную группу команду Закрыть. Сначала добавим в командную панель подчиненную ей группу кнопок. Для этого выделим в дереве элементов формы элемент Командная панель, нажмем кнопку Добавить и выберем тип группы Группа кнопок (рис. 1.202).

В дереве элементов появится новая группа, и для нее откроется окно свойств. Заполним свойства элемента формы (см. рис. 1.202):

- Имя – ГруппаЗакрыть;
- Заголовок – «Группа закрыть» (используется, когда группа имеет вид Подменю);
- Подсказка – «Команды закрытия формы».

Теперь перетащим в группу кнопку ФормаЗакрыть. А в свойстве этой кнопки Вид установим значение Кнопка командной панели.

В результате нашей настройки команда Закрыть форму выглядит как обычная кнопка командной панели. Но поскольку она помещена в группу, то в командной панели формы эта команда визуально

отделена отступами (или разделителями в подменю Еще) от остальных команд (рис. 1.203).

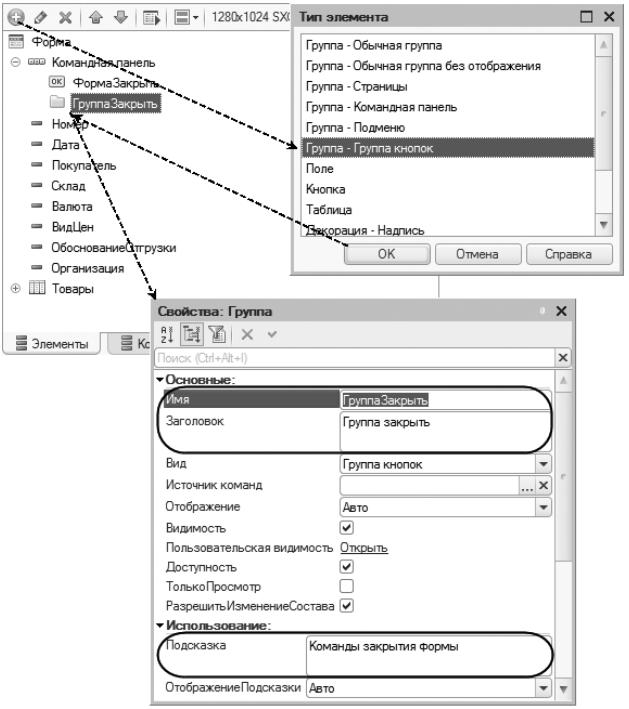


Рис. 1.202. Добавление элемента формы «Группа – Группа кнопок»

Как и для кнопки, для группы кнопок можно настраивать доступность и видимость. Настройки, сделанные для группы, будут применены ко всем командам, включенным в эту группу.

Помимо командной панели кнопку можно расположить и непосредственно в форме – подчинить узлу Форма в дереве элементов формы.

Это можно сделать так же, как и при заполнении командной панели – перетаскиванием команды. Однако перетащить команду необходимо не в узел Командная панель, а в узел Форма в дереве элементов формы.

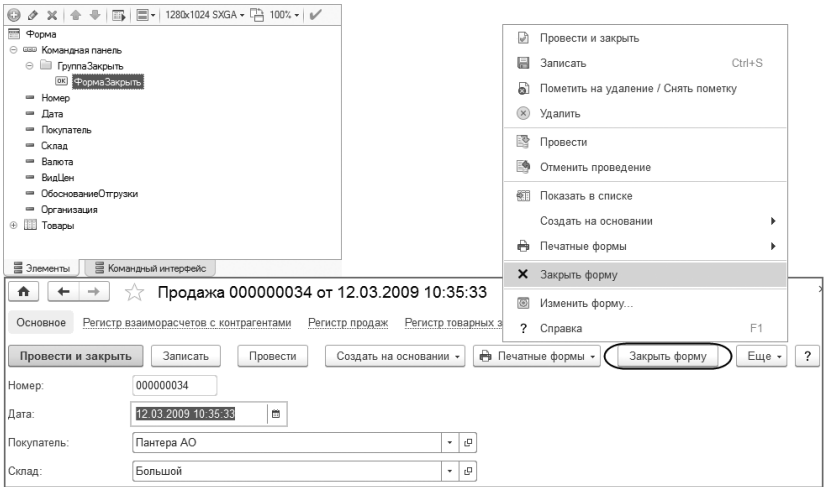


Рис. 1.203. Группа с командой «Закрыть»

Также можно воспользоваться командой Добавить (аналогично тому, как мы добавляли группу) и выбрать добавляемый элемент формы с типом Кнопка.

ВНИМАНИЕ!

В случае использования команды Добавить не происходит автоматического связывания элемента формы Кнопка с командой. Разработчик должен настроить связь самостоятельно в свойстве кнопки ИмяКоманды.

Добавим новую кнопку командой. В результате в дереве элементов появится новый элемент с типом Кнопка, подчиненный элементу Форма, и откроется окно свойств добавленного элемента (рис. 1.204).

Переместим кнопку под командную панель формы. Для этого используем команды перемещения Вверх/Вниз (или клавиатурные сокращения Ctrl + Shift + Up и Ctrl + Shift + Down).

Для кнопок, автоматически размещенных системой в командной панели, изменить порядок их следования невозможно.

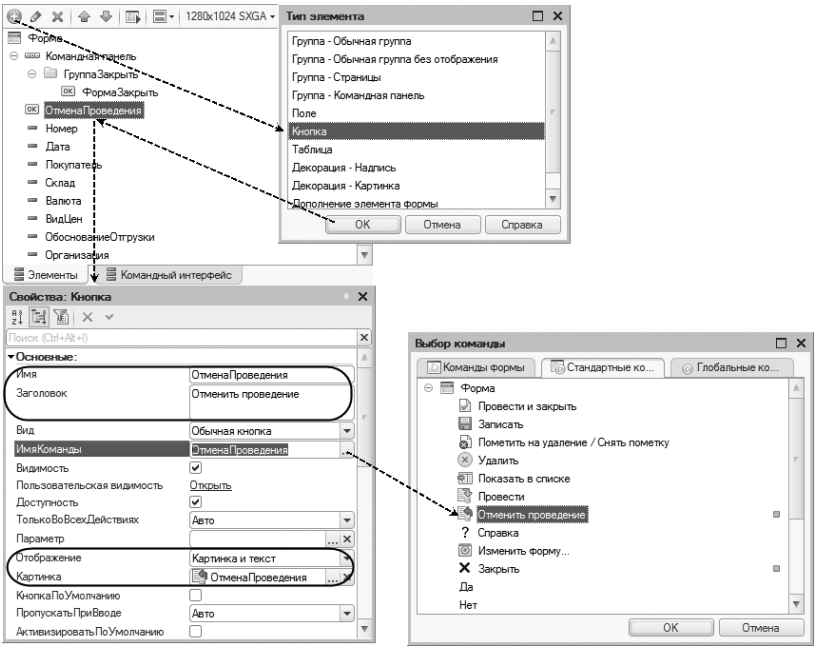


Рис. 1.204. Кнопка на форме, связанная с командой «Отмена проведения»

Заполним свойства кнопки (см. рис. 1.204):

- Имя – ОтменаПроведения;
- Заголовок – «Отменить проведение»;
- Отображение – Картинка и текст;
- Картинка – ОтменаПроведения.

Для свойства ИмяКоманды выберем в диалоге выбора команд вызываемую команду Отменить проведение.

В диалоге выбора команды можно выбрать не только стандартные команды формы, а вообще все команды, доступные в контексте формы. Для выбора команд служат соответствующие закладки диалога: Команды формы, Стандартные команды и Глобальные команды.

В результате выполненных действий в форме появилась новая кнопка, расположенная под командной панелью формы. Эту кнопку

мы добавили в форму вручную. Согласно правилам размещения стандартных команд, о которых мы рассказывали выше, в подменю Еще также была автоматически добавлена аналогичная команда Отменить проведение (рис. 1.205).

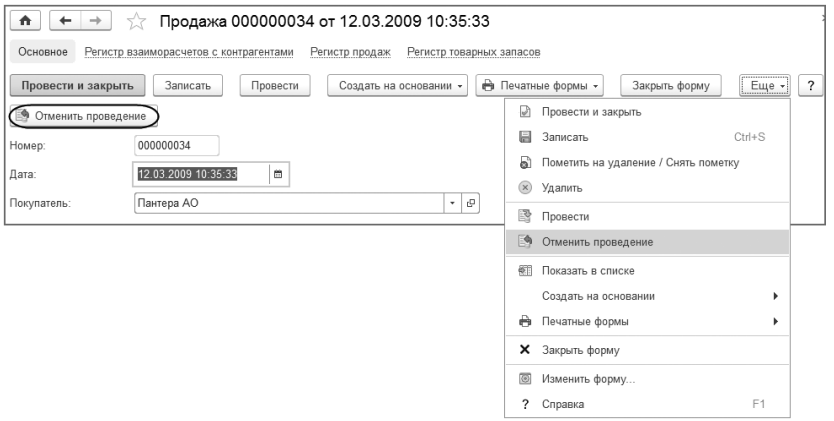


Рис. 1.205. Добавленная команда в интерфейсе

Для кнопки, расположенной в форме, применимы те же настройки, что и для кнопки командной панели.

Например, мы можем установить для кнопки зеленый цвет текста заголовка (свойство `ЦветТекста`), свойство `ПоложениеКартинки` установить в значение Право, а свойство `ОтображениеФигуры` установить в значение При активности. В результате в форме документа заголовок кнопки Отменить проведение показывается зеленым цветом, картинка отображается справа от заголовка, а сама кнопка принимает привычный выпуклый вид только при наведении на нее указателя мыши (рис. 1.206).

> Если кнопка добавляется в командную панель и с ней связывается команда, уже автоматически размещенная системой в этой командной панели, добавляемая команда замещает существующую.

На этом мы закончим рассмотрение стандартных локальных команд формы и займемся глобальными командами.

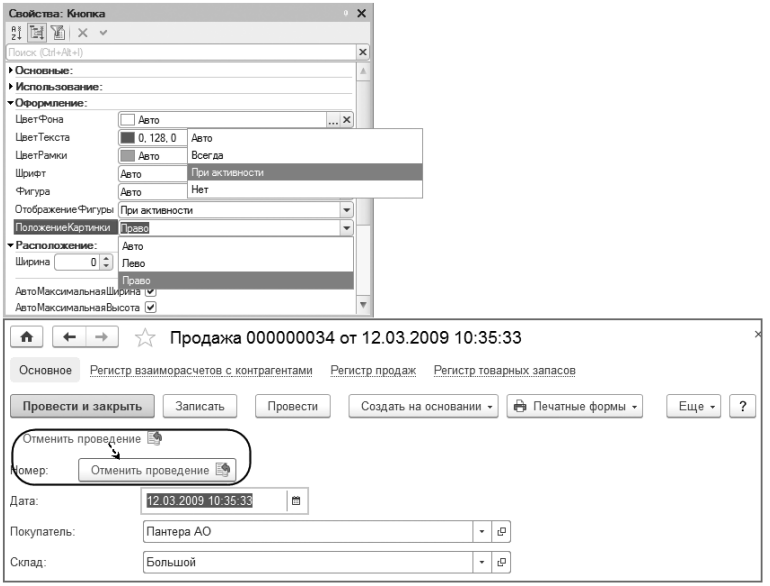


Рис. 1.206. Варианты оформления кнопки

Глобальные команды

Разработчик может расширить (по сравнению с автоматически размещенными) состав глобальных команд, доступных в форме. При этом система не ограничивает разработчика в выборе места размещения добавляемой команды.

Однако следует учитывать, что в панели навигации окна клиентского приложения рекомендуется располагать «безопасные» команды – те, которые не приводят к модификации данных, не запускают обработки и т. д. Основное назначение команд в панели навигации – сформировать контекст выполнения каких-либо действий по обработке данных.

Соответственно, в командной панели формы рекомендуется размещать команды, которые обрабатывают данные: заполняют, пересчитывают, печатают, записывают в информационную базу и т. д.

С другой стороны, система не позволит удалить из формы автоматически включенные глобальные команды. Она позволит лишь настроить их видимость и размещение.

Как и для стандартных, добавление глобальных команд проще всего выполнить перетаскиванием требуемой команды из редактора команд (закладка Глобальные команды) в редактор командного интерфейса (закладка Командный интерфейс), в узел Панель навигации или в узел Командная панель.

Например, добавим в форму глобальную параметризованную команду Печать штрихкода, которая подчинена справочнику Товары. Команда выполняет печать, поэтому добавим ее в группу Печатные формы командной панели формы (рис. 1.207).

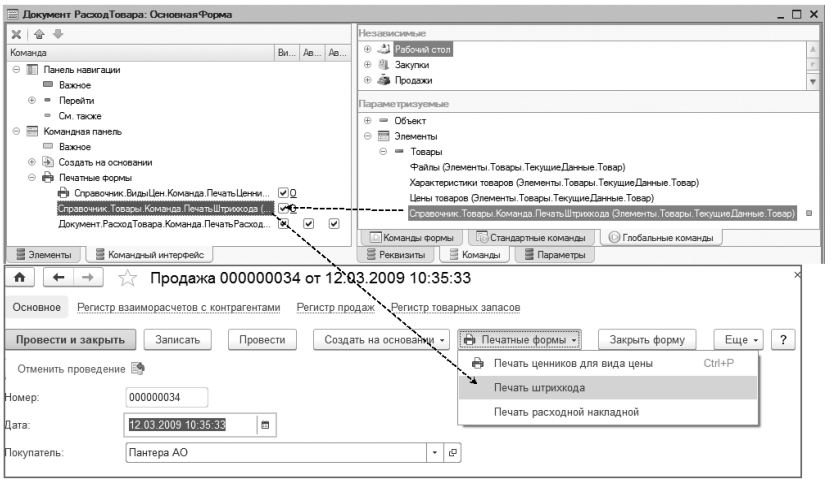


Рис. 1.207. Глобальная команда, добавленная вручную

При необходимости разработчик может настроить видимость и размещение глобальных параметризованных команд.

При этом для команд, помещенных в форму автоматически, необходимо сбросить соответствующие признаки – Автовидимость и Автоположение. В противном случае будут использованы видимость и размещение команд, формируемые системой по умолчанию.

Для команд, добавленных вручную, система не определяет признаки Автовидимость и Автозаполнение, то есть разработчик самостоятельно размещает команды и устанавливает их видимость.

Настройку размещения можно выполнить перетаскиванием команды мышью в нужную группу. Настройка видимости выполняется в окне Настройка видимости по правилам, совпадающим с настройкой видимости, показанной нами ранее для элементов формы Кнопка.

Давайте переместим команду Печать расходной накладной в группу Важное и сделаем ее невидимой по умолчанию для роли Менеджер по ценам (рис. 1.208).

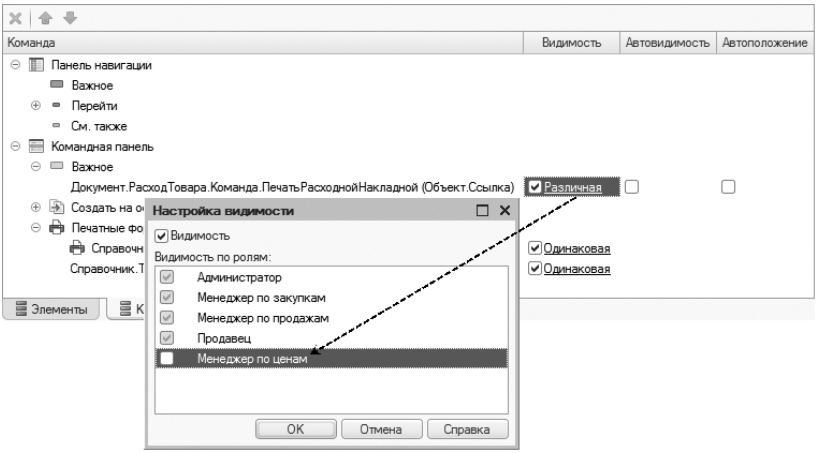


Рис. 1.208. Настройка размещения и видимости добавленной глобальной команды

В результате выполненной настройки для пользователя с ролью Менеджер по ценам команда печати невидима, а для остальных ролей – видима (рис. 1.209).

Теперь добавим в форму глобальную параметризованную команду Цены товаров. Эта команда не предназначена для обработки данных, поэтому добавим ее в панель навигации окна клиентского приложения, после всех команд панели навигации (рис. 1.210).

Настройка видимости и размещения команд в панели навигации выполняется аналогично настройке команд в командной панели. Видимость по умолчанию для команды изменять не будем – она будет видима всем ролям.

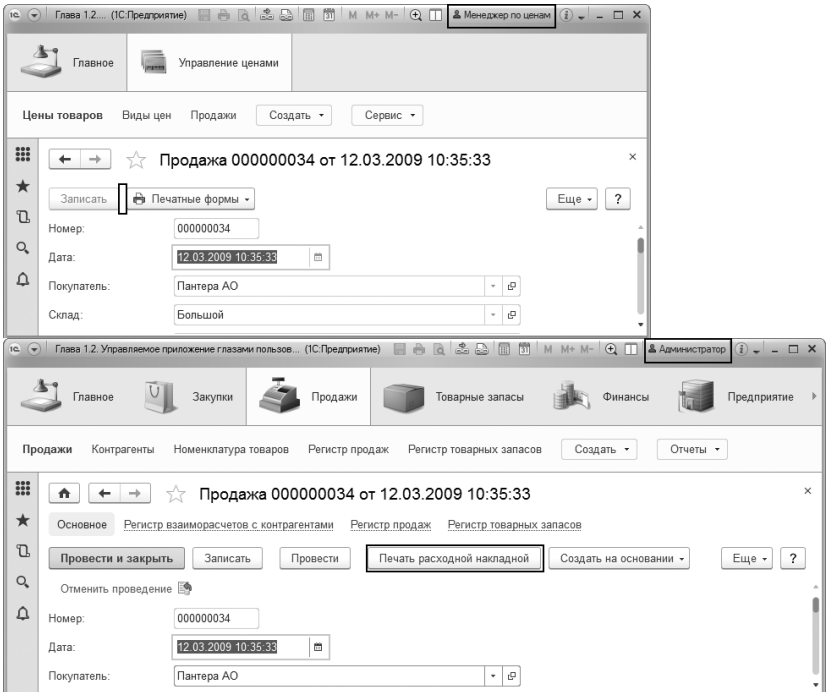


Рис. 1.209. Для разных ролей у команды «Печать расходной накладной» различная видимость по умолчанию

Обратите внимание: для роли Менеджер по ценам появилась доступная навигационная команда, причем она видима по умолчанию. Вследствие этого в форме документа РасходТовара (Продажа) отобразилась панель навигации с добавленной командой (см. рис. 1.210).

Теперь мы знаем, как сформировать набор команд формы, используя стандартные команды формы и глобальные команды командного интерфейса.

А что делать, если функциональности этих команд не хватает? Об этом мы поговорим в следующем разделе.

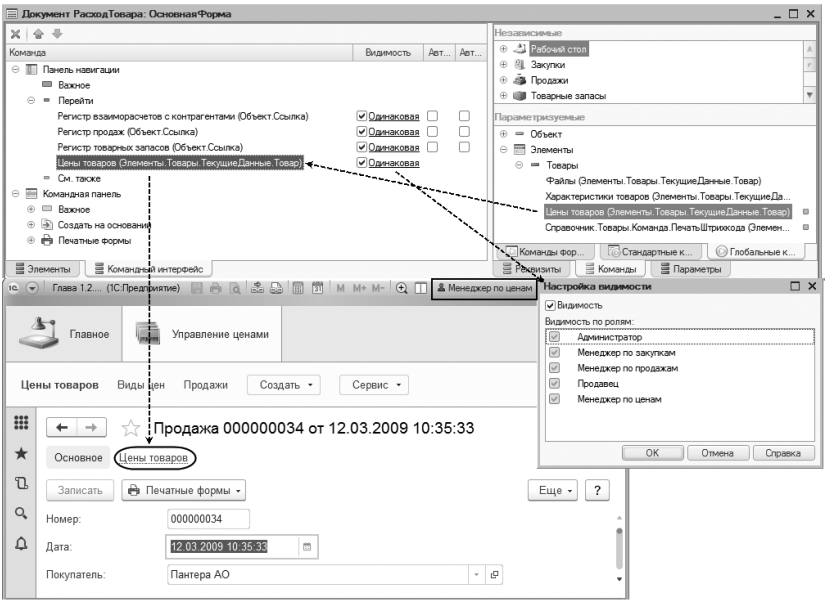


Рис. 1.210. Добавлена глобальная навигационная команда «Цены товаров»

Если не хватает стандартных команд

В случае, когда для реализации функциональных возможностей формы стандартных команд не хватает, разработчик может создать необходимое количество *произвольных команд формы*. Такие команды создаются в редакторе команд на закладке Команды формы.

Давайте создадим произвольную команду подбора товаров в документе РасходТовара.

Для создания произвольной команды воспользуемся командой Добавить на закладке Команды формы в редакторе команд. В результате в список произвольных команд формы будет добавлена новая команда и для нее откроется окно свойств (рис. 1.211).

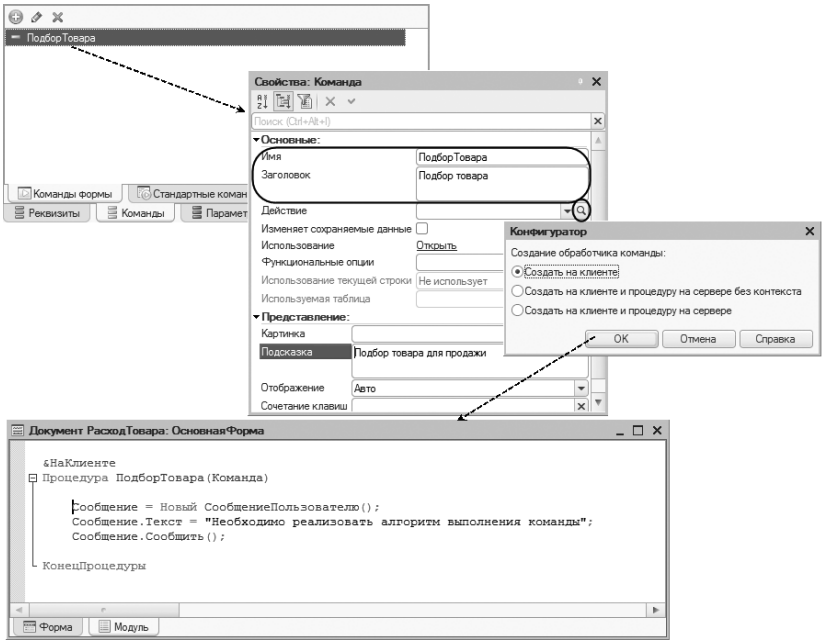


Рис. 1.211. Добавление произвольной команды формы

Заполним для команды свойства (рис. 1.211):

- Имя – ПодборТовара;
- Заголовок – «Подбор товара»;
- Подсказка – «Подбор товара для продажи».

Для произвольной команды формы необходимо написать программный код исполнения.

Как и для произвольных глобальных команд, сам функционал команды мы реализовывать не будем.

Процедура исполнения команды располагается в модуле формы. Используя свойство Действие произвольной команды, необходимо связать команду с процедурой исполнения.

Для этого нужно нажать кнопку открытия (со значком лупы) справа от поля выбора свойства Действия. После этого необходимо выбрать

тип обработчика команды. Выберем значение по умолчанию Создать на клиенте. В результате в модуле формы будет создан шаблон процедуры – обработчика команды, исполняющейся на клиенте. И эта процедура автоматически будет указана в качестве обработчика команды в свойстве Действия (ПодборТовара). Разработчику останется только написать программный код исполнения команды (в нашем случае это будет просто сообщение пользователю).

Для предоставления пользователю доступа к созданной команде последнюю необходимо связать с элементом формы типа Кнопка – аналогично тому, как мы это делали для стандартных команд формы.

Перетащим команду в окно элементов формы в командную панель таблицы Товары. Кроме того, зададим свойства для оформления новой кнопки в интерфейсе. Установим желтый цвет фона кнопки (свойство ЦветФона), отображение заголовка кнопки курсивом (свойство Шрифт), свойство Фигура в значение Овальная.

В результате пользователь получил возможность выбора команды Подбор товара из командной панели таблицы товаров (рис. 1.212).

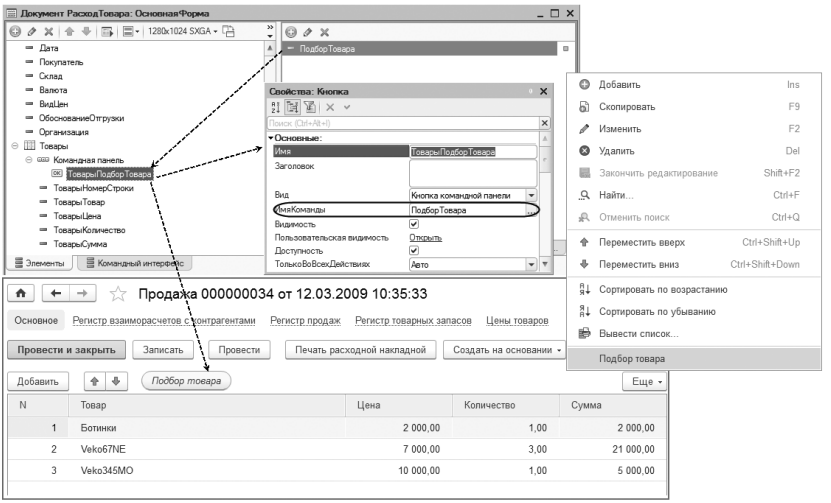


Рис. 1.212. Произвольная команда формы добавлена в интерфейс

Настройка размещения и видимости произвольных команд выполняется аналогично настройке стандартных команд – и стандартная, и произвольная команды связываются с элементом формы одного и того же типа Кнопка. А доступность и видимость фактически настраиваются уже для этого элемента.

Кроме настройки доступности элемента формы, для произвольной команды разработчик может настроить ее *доступность в разрезе ролей* с помощью свойства Использование самой команды. Настройка выполняется в диалоговом окне Настройка использования (рис. 1.213).

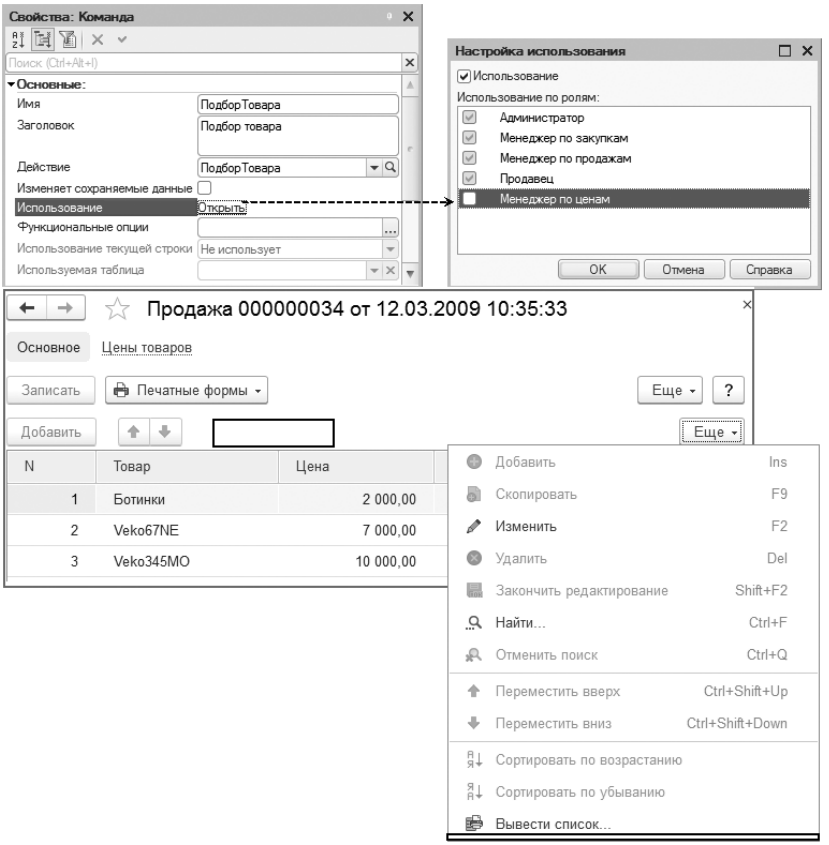


Рис. 1.213. Настройка доступности произвольной команды в разрезе ролей

При этом разработчик может определить:

- Общую доступность команды – если флажок Использование в диалоговом окне установлен, то команда доступна всем ролям с «третьим» значением флажка.
- Доступность команды для конкретной роли пользователя – если рядом с ролью в списке Использование по ролям флажок установлен, то команда доступна для роли; если сброшен, то недоступна; если же флажок установлен в «третье» значение (на сером фоне), то для роли используется значение общей доступности.

Например, для пользователя с ролью Менеджер по ценам команда Подбор товара недоступна (флажок сброшен), см. рис. 1.213. Для этого пользователя в командной панели таблицы Товары команды нет.

Доступность произвольных команд также можно поставить в зависимость от значений функциональных опций. Для такой настройки используется свойство команды Функциональные опции.

Например, зададим зависимость команды Подбор товара от значения функциональной опции Работа с торговым оборудованием (рис. 1.214).

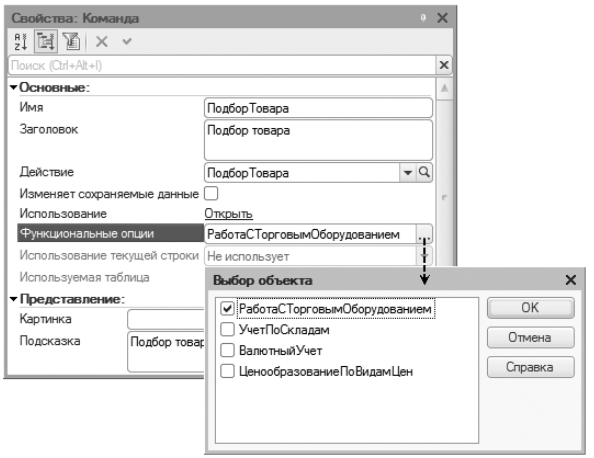


Рис. 1.214. Настройка доступности произвольной команды в зависимости от функциональной опции

Для одного и того же пользователя при истинном значении функциональной опции команда доступна, а при ложном – недоступна (рис. 1.215).

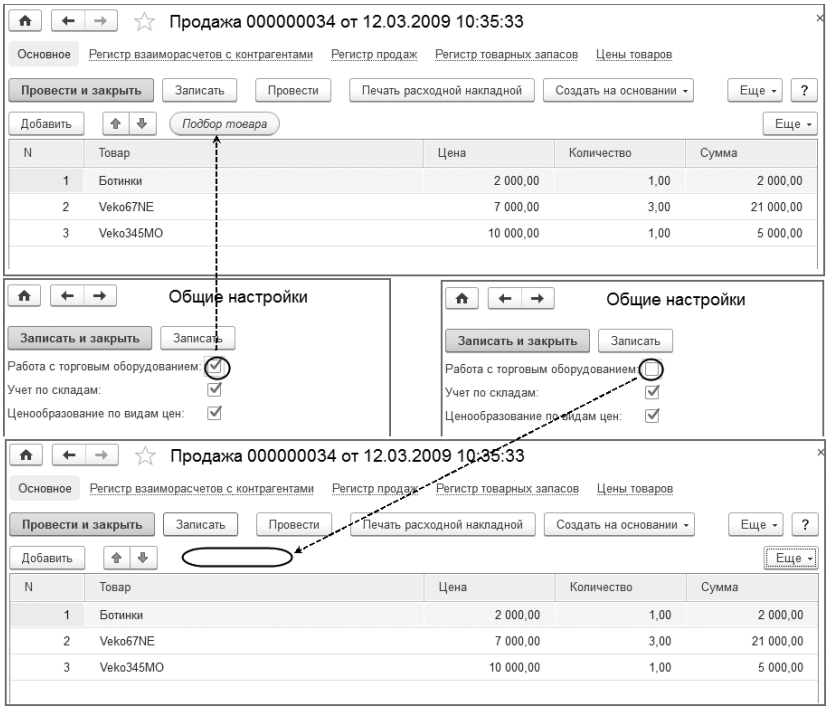


Рис. 1.215. Зависимость доступности команды от значения функциональной опции

ВНИМАНИЕ!

Свойства команды Доступность и Функциональные опции имеют приоритет перед свойствами элемента управления Кнопка.

Краткие итоги

Все команды, которые могут присутствовать в форме, можно разделить на два класса (рис. 1.216):

- Локальные команды формы – обеспечивают доступ к функциональности формы.
- Глобальные команды – обеспечивают доступ к функциональности приложения.

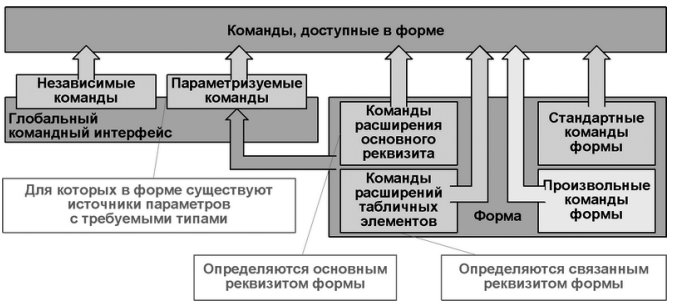


Рис. 1.216. Классификация команд формы

В состав команд формы входят:

- Стандартные команды формы – обеспечивают работу с формой.
- Стандартные команды расширения основного реквизита формы – обеспечивают обработку данных формы.
- Стандартные команды расширений некоторых элементов формы – обеспечивают обработку данных формы.
- Произвольные команды формы – назначение команды определяется разработчиком.

Глобальные команды, доступные в форме, включают в себя:

- Независимые глобальные команды.
- Часть параметризованных глобальных команд – те, для которых в форме существует источник параметра требуемого типа.

При разработке командного интерфейса формы разработчик должен ответить на вопросы:

- какие действия нужно будет выполнять с данными – для обработки данных используются команды формы;
- какие дополнительные данные (помимо данных формы) должны быть доступны и какие дополнительные действия (помимо обработки данных формы) необходимо выполнять – для обеспечения этой функциональности используется подмножество глобальных команд;
- каким пользователям будут доступны команды – выполнить ролевую настройку доступности;
- как должна выглядеть форма – настроить размещение и видимость команд.

Для редактирования состава команд формы, их расположения и видимости используется редактор формы. В редакторе формы описывается состав команд командной панели формы и состав команд панели навигации формы.

Часть команд в командную панель и в панель навигации помещается системой автоматически. В редакторе форм разработчик эти команды удалить не может, но он может настроить видимость этих команд в разрезе ролей пользователей.

Для того чтобы пользователь смог вызвать команду, она должна быть связана с элементом формы Кнопка (закладка Элементы редактора форм) или помещена в командный интерфейс (закладка Командный интерфейс редактора форм).

ЧАСТЬ 2

Конструирование форм

Оглавление

Глава 2.1. Что такое форма

Концепция построения форм

В данном разделе будет рассматриваться вопрос создания и конструирования форм в режиме Конфигуратор.

Форма является важнейшим связующим звеном в цепи «пользователь – данные». Именно в формах мы редактируем данные, вводим новую информацию, видим результаты работы.

Формы «1С:Предприятия» наделены различными возможностями как построения, так и отображения данных. В большинстве случаев от разработчика не требуется заботиться о том, как на экране будет выглядеть тот или иной элемент, система сама сможет позаботиться об этом. От разработчика требуется правильно настроить интерфейсные свойства объекта конфигурации, на основании которых и будет построено нужное представление.

Управляя размещением элементов в форме, разработчик должен «посоветовать» системе те или иные способы группировки элементов, порядок их размещения. Естественно, платформа предоставляет возможности взять под контроль некоторые этапы разработки формы, но данное действие не является приоритетным. Задача разработчика состоит не в детальном, «попиксельном», размещении элементов на форме, не в описании сложных привязок, а в логическом описании состава формы.

Необходимо описать состав формы в виде дерева элементов, которые будут отображать данные, добавить в описание необходимые реквизиты и команды. Скомпоновать элементы в логические группы, определить порядок обхода элементов формы.

Полученное от разработчика описание формы, другие факторы, влияющие на внешний вид и функциональность формы, помогут системе построить форму на экране и тем самым освободить ресурсы разработчика не для «рисования», а для разработки функциональности конкретного прикладного решения.

Разработчик может влиять на расположение и внешний вид элементов формы. Для этого у него в руках инструмент свойств элементов формы. Существует возможность перенастройки командных панелей формы, дополнительных кнопок у элементов формы, объединения элементов в группы, распределения их по страницам, настройки колонок списков. Однако все эти возможности призваны лишь помочь системе в построении формы, а не полностью заменить это построение ручным способом.

Помимо простого открытия форм для просмотра или редактирования данных существует возможность открытия с установленным отбором, с выделением каких-либо конкретных данных из общего числа. Помочь в этом могут, например, программная установка отборов и параметризуемые команды.

При построении форм системой учитываются не только настройки самой формы, сделанные разработчиком. Влияние на поведение формы и ее элементов оказывают настройки прав пользователей, применяемые к сеансу работы приложения функциональные опции, настройки, которые сделал сам пользователь в сеансе своей работы.

Отдельно стоит упомянуть о возможностях форм сообщать пользователю об ошибках, возникающих в процессе работы. Например, это может быть сообщение о незаполненном поле, данные в которое должны быть внесены обязательно. Система сообщит о таком типе ошибки, выделив и активизировав именно этот элемент.

Формы как элемент общения программы с пользователем

Помимо «стандартных» форм различных объектов конфигурации (справочников, документов, регистров и пр.) существует одна особенная форма – *начальная страница*. Именно с нее и начинается работа пользователя с системой.

Можно сказать, что начальная страница призвана быть лицом разрабатываемого приложения, его визитной карточкой. Именно на начальную страницу необходимо при проектировании интерфейса приложения выносить всю необходимую в первую очередь информацию. Это могут быть списки наиболее часто используемых документов, справочников, данные отчетов, формы обработок.

Сказать, что на начальной странице отображаются просто данные, неверно. На ней отображаются формы других объектов конфигурации. И, как уже было сказано ранее, на начальную страницу оказывает влияние все то, что способно повлиять на любую форму прикладного решения: права доступа, функциональные опции, настройки пользователя. Именно отображение других форм и выделяет начальную страницу из всех форм конфигурации. Подробнее о настройке начальной страницы будет рассказано в главе 2.10 на стр. 422.

Для формы может быть назначен основной реквизит, который частично определяет поведение формы, то, какие данные в такой форме можно просматривать, редактировать, как будет вести себя форма в той или иной ситуации.

При работе в интерфейсе «Такси», который рассматривается в книге, большинство форм открывается в *окнах клиентского приложения*, которые расположены в рабочей области основного окна приложения. Этих окон может быть открыто сколько угодно, но, поскольку интерфейс «Такси» – однооконный, каждое следующее окно замещает предыдущее. Поэтому в каждый конкретный момент пользователь видит перед собой только одно окно.

Исключением являются *блокирующие окна*, которые могут блокировать как окно владельца, так и сразу весь интерфейс приложения. Блокирующие окна применяются для вспомогательных задач (например, ввода и выбора каких-либо данных) или для интерактивного общения с пользователем.

«1С:Предприятие» позволяет вообще не описывать форму в конфигураторе. При обращении к объекту в пользовательском режиме система сгенерирует необходимую форму самостоятельно. Если разработчиком прикладного решения не планируется вносить изменения в работу и поведение формы (например, менять порядок обхода элементов, добавлять команды, описывать с помощью встроенного языка алгоритмы заполнения данных), то создавать форму в режиме Конфигуратор необязательно.

Среда существования формы

Форма существует и на сервере, и на клиенте. Связь элементов формы с данными информационной базы осуществляется с помощью *реквизитов формы* (данные формы). Все данные, которые планируется отображать или редактировать в форме, должны быть обязательно описаны в виде реквизитов.

Процесс создания и открытия формы, отображающей объектные данные (у формы определен основной реквизит), выглядит так:

- Объект считывается из базы данных на сервере.
- Объект конвертируется в данные формы (создание формы на сервере).
- Объект удаляется из памяти.
- Данные формы передаются на клиент (создание формы на клиенте).

Запись данных формы в информационную базу происходит только на сервере:

- Данные формы получаются с клиента.
- Данные формы конвертируются в объект.
- Объект записывается в базу данных.
- Объект удаляется из памяти.

Графически все вышесказанное можно изобразить в виде схемы (рис. 2.1).

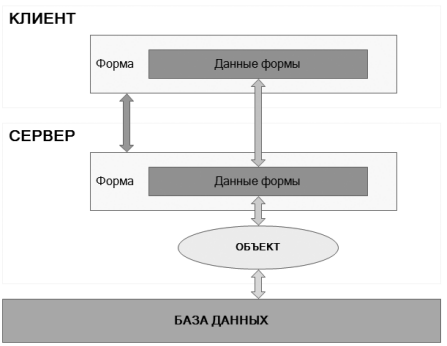


Рис. 2.1. Взаимодействие формы и базы данных

В процессе своего создания форма проходит через определенный фильтр различных факторов (рис. 2.2), влияющих на ее внешний вид. Это совокупность ролей (прав доступа), назначенных пользователю разработчиком прикладного решения, функциональные опции приложения, настройки, сделанные пользователем в предыдущем и текущем сеансе работы.

Рис. 2.2. Процесс открытия формы

Между клиентом и сервером происходит обмен не только данными формы, но и ее оформительскими свойствами. К оформительским относятся все свойства формы, влияющие на ее внешний вид. Таким образом достигается полное соответствие внешнего вида формы и на клиенте, и на сервере.

Глава 2.2. Создание формы

Создание формы с помощью конструктора

Создание новой формы начинается с *конструктора форм*. Модификация формы, созданной конструктором, осуществляется в *редакторе формы* и описана в главе 2.3 на стр. 256.

В конфигурации могут существовать формы, подчиненные объектам конфигурации, и общие формы. Общие формы располагаются в дереве конфигурации, в ветке Общие – Общие формы. Конструктор форм различает, какая форма будет создаваться: подчиненная объекту или общая.

При создании общей формы конструктор предлагает выбрать тип формы из следующих типов (рис. 2.3):

- Произвольная форма;
- Форма констант;
- Форма отчета;
- Форма настроек отчета;
- Форма варианта отчета;
- Форма настроек динамического списка;
- Форма поиска.

При выборе произвольной формы разработчик самостоятельно в процессе разработки будет определять данные, с которыми будет взаимодействовать форма.

При выборе остальных типов форм разработчик указывает тип данных, с которыми в дальнейшем будет взаимодействовать форма. В зависимости от этого конструктор автоматически добавляет в форму соответствующие данные и интерфейсные элементы, сокращая дальнейшие действия разработчика.

По умолчанию конструктор общих форм предлагает создать произвольную форму. Связано это с тем, что в подавляющем большинстве случаев разработчик будет создавать именно произвольные общие формы.

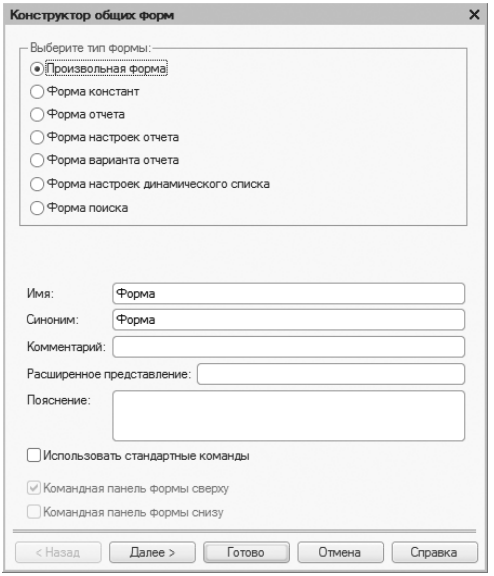


Рис. 2.3. Выбор типа общей формы

Флажок Использовать стандартные команды в конструкторе общих форм позволяет автоматически добавить команды для открытия общей формы в интерфейс той подсистемы, в состав которой она будет входить.

Существует несколько способов создать форму, подчиненную какому-либо объекту конфигурации:

- Воспользоваться кнопкой Открыть (рис. 2.4) в палитре свойств объекта конфигурации у нужного типа формы.
- Выделить в дереве конфигурации ветвь Формы объекта, для которого будет создаваться форма, и воспользоваться либо контекстным меню, либо кнопкой Добавить на панели дерева конфигурации (рис. 2.5).

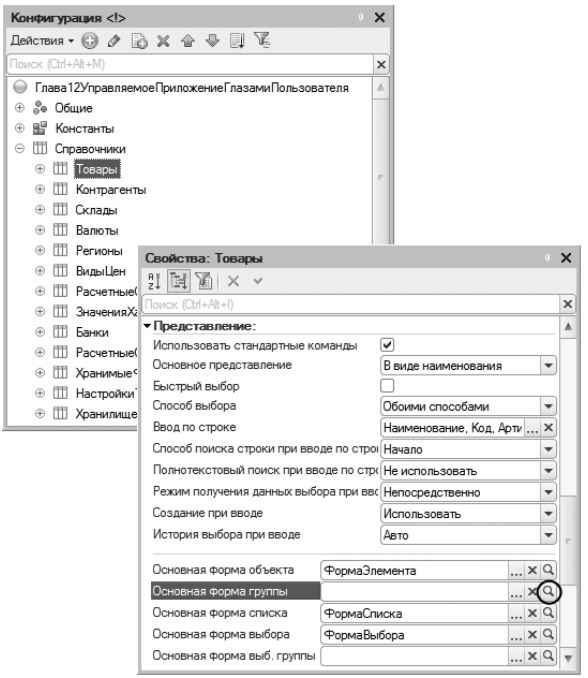


Рис. 2.4. Создание формы кнопкой «Открыть»

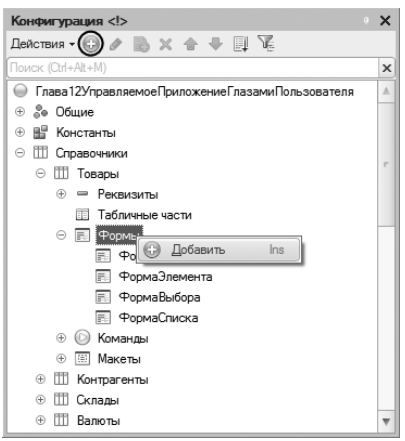


Рис. 2.5. Создание формы кнопкой «Добавить»

- Открыть окно редактирования объекта конфигурации, перейти на закладку Формы и воспользоваться либо кнопкой Открыть у необходимого типа формы, либо кнопкой командной панели Добавить, либо контекстным меню в списке форм (рис. 2.6).

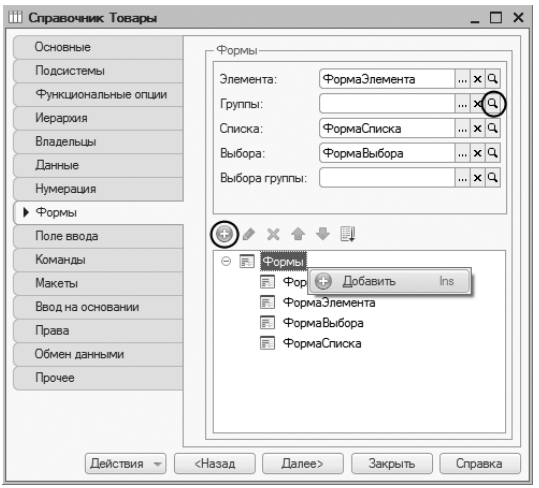


Рис. 2.6. Создание формы из окна редактирования объекта

При использовании любого из вышеперечисленных способов будет открыто окно конструктора формы объекта конфигурации. В зависимости от вида объекта, его свойств конструктором будут предложены характерные для текущего режима создания типы форм:

- Для объекта Справочник это будет форма списка, форма выбора, форма элемента. Если справочник иерархический, то возможно создание формы группы и формы выбора группы (рис. 2.7).

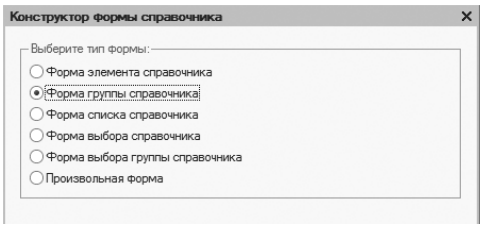


Рис. 2.7. Типы форм объекта «Справочник»

- Для объекта Документ – это форма списка, форма выбора, форма документа (рис. 2.8).

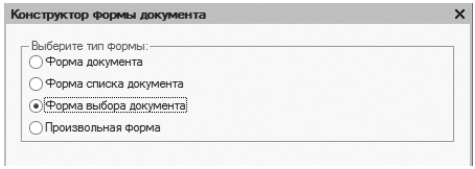


Рис. 2.8. Типы форм объекта «Документ»

- Для объекта Журнал документов – это форма журнала (рис. 2.9).

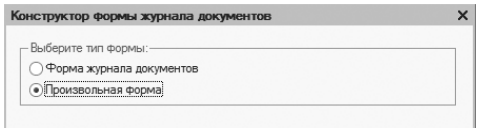


Рис. 2.9. Типы форм объекта «Журнал документов»

- Для объекта Перечисление – это форма списка и форма выбора (рис. 2.10).

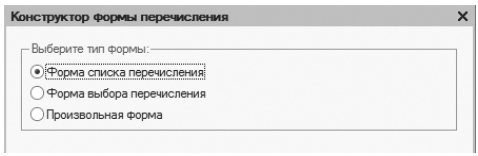


Рис. 2.10. Типы форм объекта «Перечисление»

- Для объекта Отчет – это форма отчета, форма варианта, форма настроек (рис. 2.11).

Рис. 2.11. Типы форм объекта «Отчет»

- Для объекта Обработка это форма обработки (рис. 2.12).

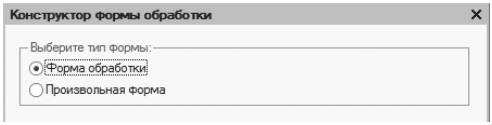


Рис. 2.12. Типы форм объекта «Обработка»

- Для объекта План видов характеристик – это форма списка, форма выбора, форма элемента. Если план видов характеристик иерархический, то возможно создание формы группы и форма выбора группы. Если объект неиерархический, то создание форм группы недоступно (рис. 2.13).

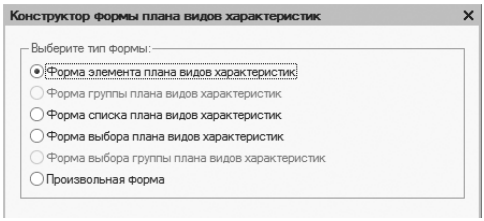


Рис. 2.13. Типы форм объекта «План видов характеристик»

- Для объекта План счетов – это форма списка, форма выбора, форма счета (рис. 2.14).

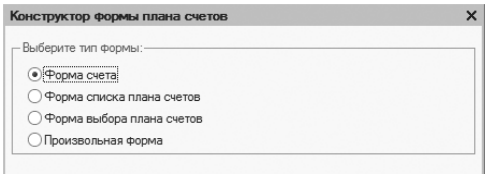


Рис. 2.14. Типы форм объекта «План счетов»

- Для объекта План видов расчета – это форма списка, форма выбора, форма вида (рис. 2.15).

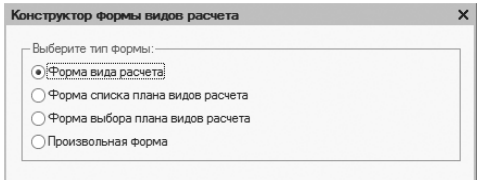


Рис. 2.15. Типы форм объекта «План видов расчета»

- Для объекта Регистр сведений – это форма списка, форма записи, форма набора записей (рис. 2.16).

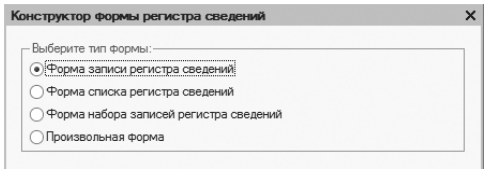


Рис. 2.16. Типы форм объекта «Регистр сведений»

- Для объекта Регистр накопления – это форма списка и форма набора записей (рис. 2.17).

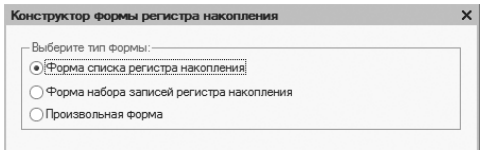


Рис. 2.17. Типы форм объекта «Регистр накопления»

- Для объекта Регистр бухгалтерии – это форма списка и форма набора записей (рис. 2.18).

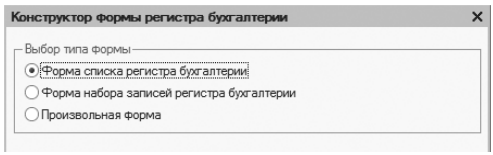


Рис. 2.18. Типы форм объекта «Регистр бухгалтерии»

- Для объекта Регистр расчета – это форма списка и форма набора записей (рис. 2.19).

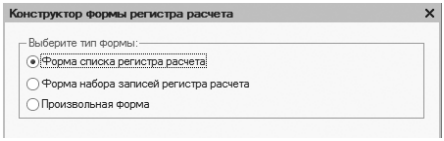


Рис. 2.19. Типы форм объекта «Регистр расчета»

- Для объекта Бизнес-процесс – это форма списка, форма выбора, форма бизнес-процесса (рис. 2.20).

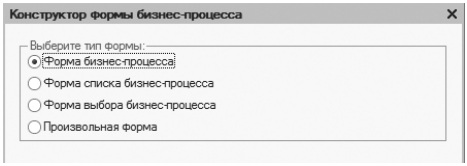


Рис. 2.20. Типы форм объекта «Бизнес-процесс»

- Для объекта Задача – это форма списка, форма выбора, форма задачи (рис. 2.21).

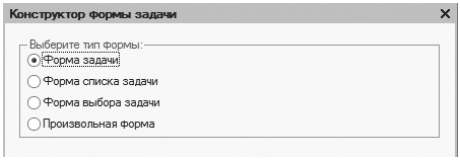


Рис. 2.21. Типы форм объекта «Задача»

- Для объекта Критерий отбора – это форма критерия отбора (рис. 2.22).

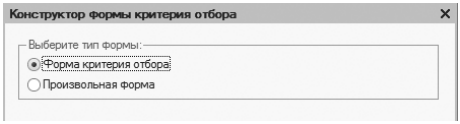


Рис. 2.22. Типы форм объекта «Критерий отбора»

- Для объекта План обмена – это форма списка, форма выбора и форма узла (рис. 2.23).

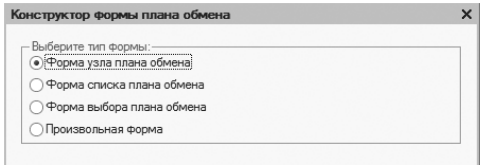


Рис. 2.23. Типы форм объекта «План обмена»

- Для объекта Хранилище настроек – это форма сохранения и форма загрузки настроек (рис. 2.24).

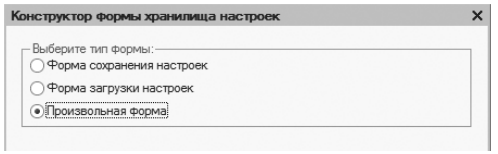


Рис. 2.24. Типы форм объекта «Хранилище настроек»

Кроме всех перечисленных форм для любого из объектов конфигурации можно создать тип формы Произвольная. Такая форма после создания не связана ни с какими данными конфигурации. Действия, которые можно будет выполнять в такой форме, целиком и полностью будут зависеть от желания и способностей разработчика формы.

Формы, связанные через свой основной реквизит с объектами конфигурации, уже наделены определенными свойствами, характеристиками, методами. Состав таких свойств, характеристик, методов зависит от объекта конфигурации.

Например, форму редактирования констант можно создать с помощью контекстного меню ветви конфигурации Константы (рис. 2.25). Созданная таким образом форма будет размещаться в ветви конфигурации Общие формы.

При выборе того или иного типа создаваемой формы конструктор форм автоматически формирует Имя и Синоним формы. При формировании учитывается наличие у объекта конфигурации

формы с таким именем. Если такая форма существует, то к имени создаваемой формы добавится числовой показатель (рис. 2.26).

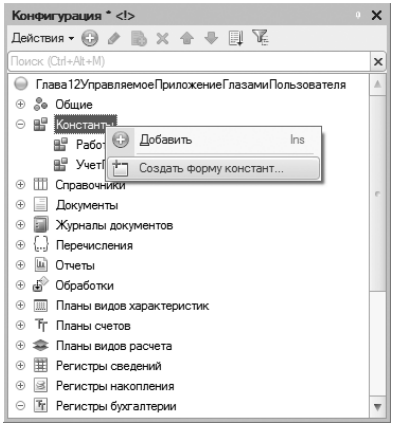


Рис. 2.25. Создание формы констант

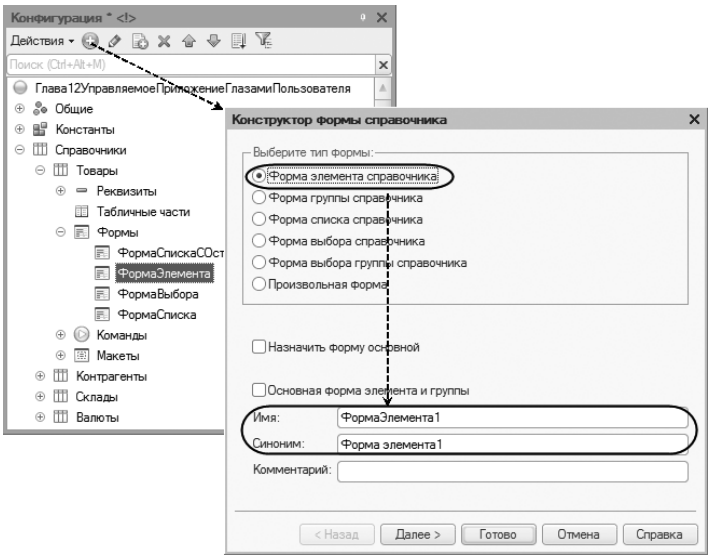


Рис. 2.26. Формирование имени формы

Если разработчика не устраивает имя формы, которое предлагает конструктор, то его можно изменить. При этом возможно автоматическое формирование и нового синонима формы. Синоним формы – имя, под которым форма может фигурировать в интерфейсе пользователей прикладного решения. Синоним формы также можно изменить.

При необходимости можно определить создаваемую форму в качестве основной формы для объекта конфигурации. Основная форма – это та форма, которая будет открываться стандартными командами «1С:Предприятия», размещенными в интерфейсе. Если разработчик хочет открыть форму, не назначенную основной, то для этого ему нужно создать собственную команду, которая будет открывать эту форму.

Чтобы назначить форму основной, в конструкторе форм необходимо установить соответствующий флажок (рис. 2.27). Если для объекта конфигурации создаваемый тип форм еще не создавался (нет формы создаваемого типа), то флажок Назначить форму основной устанавливается автоматически.

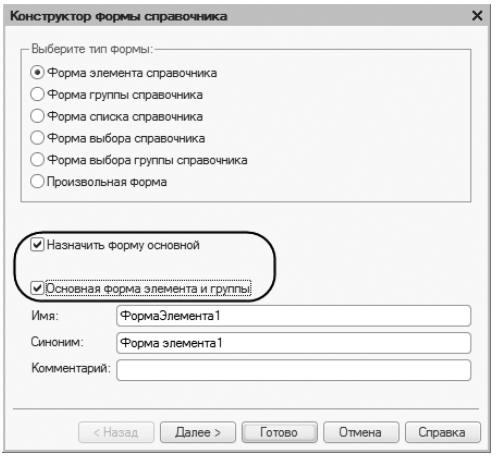


Рис. 2.27. Создание основной формы

В дальнейшем основную форму объекта можно переопределить. Как это сделать, будет рассказано в главе, посвященной редактору формы.

Для некоторых типов форм конструктор предоставляет возможность использовать их одновременно в качестве формы элемента и формы группы (см. рис. 2.27). Это удобно в тех случаях, когда и элемент, и группа имеют одинаковый состав реквизитов и одинаковые алгоритмы их заполнения; не нужно создавать две разные формы.

На этом работу с конструктором форм можно закончить (кнопка Готово) или перейти к следующему шагу (кнопка Далее), который позволит продолжить конструирование формы. На этом шаге разработчик дает указания конструктору форм на необходимость использования реквизитов объекта (или состава констант для формы констант) в создаваемой форме. Делается это с помощью установки соответствующего флажка. Здесь же можно определить, сколько колонок с элементами формы будет применяться для отображения данных (рис. 2.28).

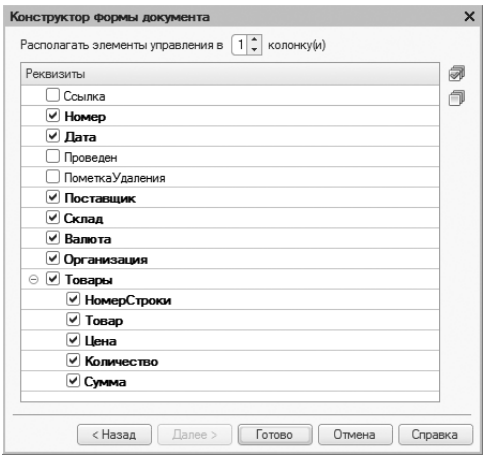


Рис. 2.28. Определение состава формы в конструкторе форм

ВНИМАНИЕ!

Управлять порядком следования элементов разрабатываемой формы в данном окне конструктора формы разработчик не может. Это можно сделать либо предварительно (определив порядок реквизитов объекта), либо после окончания работы конструктора (в окне редактора формы).

Все настройки, сделанные на втором шаге в конструкторе форм, можно потом изменить, поэтому сильно задумываться, например, о количестве колонок элементов, если вы еще не представляете в голове внешнего вида формы, не следует.

Отказаться от создания новой формы можно на любом этапе работы конструктора форм. Для этого необходимо или закрыть окно стандартным для всех окон операционной системы способом, или нажать кнопку Отмена.

Создание формы методом копирования

Кроме создания форм с помощью конструктора разработчику прикладного решения доступен вариант создания формы с помощью метода копирования.

Через буфер обмена операционной системы можно скопировать формы как текущей, так и любой произвольной конфигурации. При этом если платформа не может правильно сопоставить какие-либо свойства формы, свойства ее реквизитов, типы значений и прочее, то разработчик получит сообщение об ошибке: «Обнаружены неразрешимые ссылки».

Кроме копирования с использованием буфера обмена операционной системы также доступен метод перетаскивания форм с помощью мыши. Необходимо лишь захватить нужную форму и перетащить в новое место.

Копируемые формы можно подчинять другим объектам конфигурации. Так, например, мы можем скопировать форму некоего объекта в другой объект. Если в объекте, куда осуществляется копирование, уже существует форма с таким именем, то новой форме будет присвоено имя, к которому добавится порядковый номер (рис. 2.29).

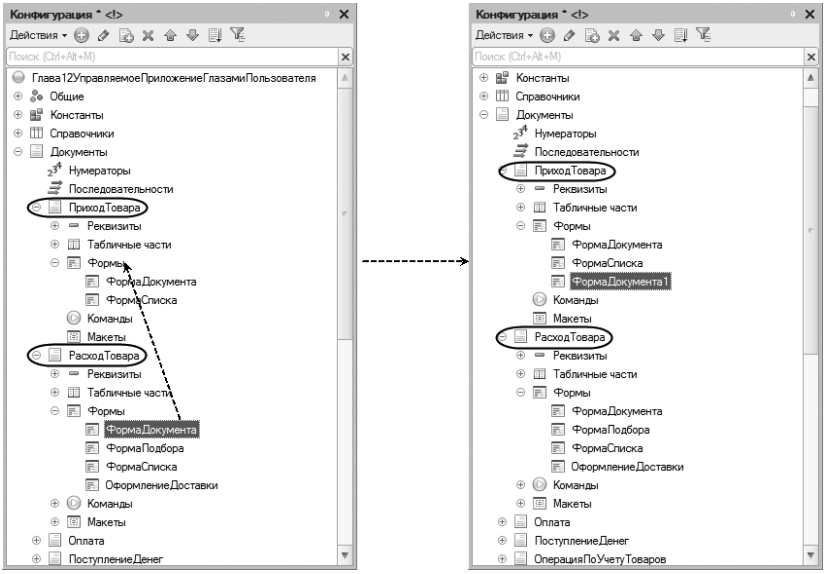


Рис. 2.29. Формирование имени формы при копировании

ВНИМАНИЕ!

При копировании формы происходит только лишь ее переподчинение в дереве конфигурации. Назначение формы, данные, с которыми она взаимодействует, при копировании не изменяются. Это означает, что, например, если скопировать форму справочника в другой справочник, то форма по-прежнему будет работать с данными первого справочника.

Глава 2.3. Редактирование формы

Описание редактора формы

Для редактирования формы в конфигураторе используется специализированный редактор форм. Окно редактора форм разбито на несколько областей (рис. 2.30), каждая из которых отвечает за ту или иную функциональность будущей формы.

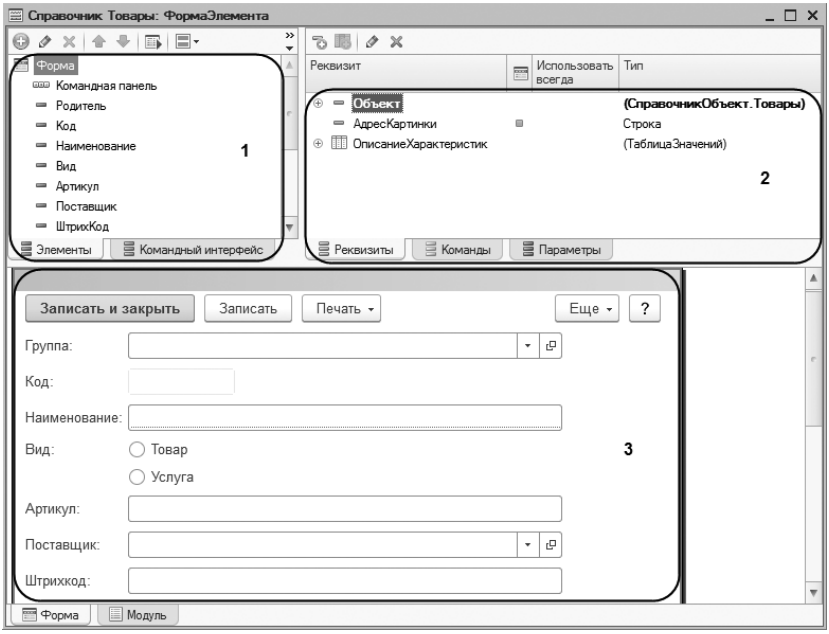


Рис. 2.30. Окно редактора формы

1. Описываются состав и порядок элементов, из которых состоит форма, команды интерфейса, которые могут выполняться в форме.
2. Описывается состав реквизитов и параметров формы, а также команды, выполняемые внутри формы.

3. Представлен внешний вид формы, как она может выглядеть на экране пользователя, с учетом описанных реквизитов, элементов, команд формы.

При изменении каких-либо настроек в окнах редактора они тут же применяются и изменяют вид формы. При выделении какого-либо элемента в окне элементов формы (1) он сразу же выделяется и показывается в окне предварительного просмотра формы (3), и наоборот.

Программный модуль описывает работу формы на встроенном языке. Для редактирования модуля формы можно воспользоваться закладкой Модуль, которая расположена внизу редактора форм.

В модуле формы располагаются обработчики событий формы, элементов формы, команд формы. Помимо предопределенных обработчиков событий разработчик прикладного решения может создавать в модуле формы свои процедуры и функции (рис. 2.31).

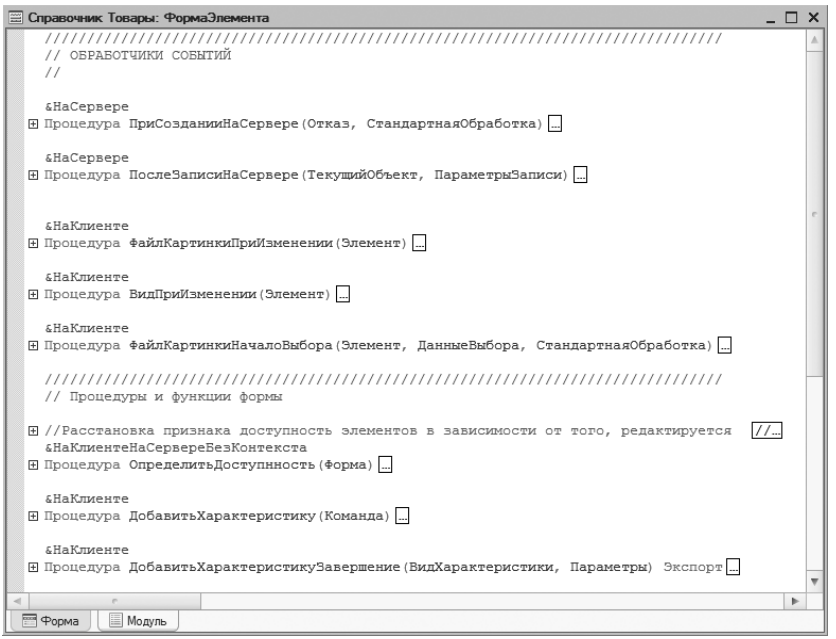


Рис. 2.31. Модуль формы

Модуль формы, так же как и другие модули конфигурации, редактируется в специализированном редакторе, в котором для удобства разработчика предусмотрены различные интерфейсные команды и меню (рис. 2.32).

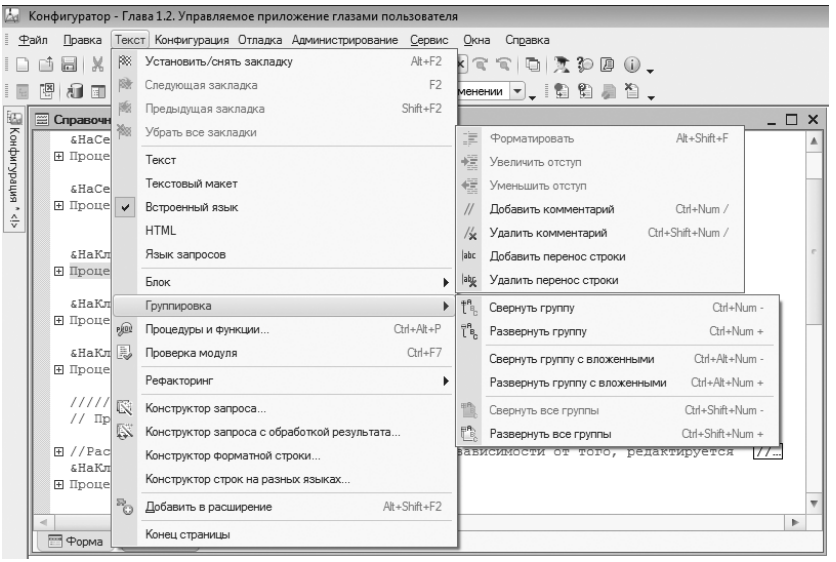


Рис. 2.32. Редактор модуля формы

С помощью закладки Параметры редактора формы можно попасть в окно редактирования параметров формы (рис. 2.33).

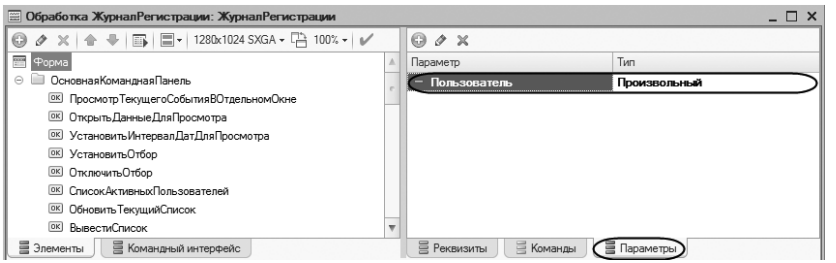


Рис. 2.33. Редактирование параметров формы

С помощью параметров формы организовано управление функциональностью формы при ее открытии в пользовательском режиме работы приложения. Если это необходимо, разработчик может изменить параметры формы уже в процессе ее работы.

Кроме параметров, описанных разработчиком формы, при ее вызове и работе доступны параметры, автоматически предоставляемые расширением формы. Для разных объектов конфигурации, определяющих основной реквизит формы, набор параметров различается.

На состав параметров формы, предоставляемых основным реквизитом формы, влияет окружение объекта конфигурации, его свойства. Примером может служить параметр Основание, если это форма документа и этот документ может вводиться на основании какого-либо объекта.

Подробнее об использовании параметров формы можно прочитать в главе 3.2 на стр. 444 и в главе 3.3 на стр. 446.

Создание, изменение и удаление элементов формы, команд, реквизитов и прочее осуществляются с помощью кнопок командных панелей соответствующих областей редактора формы. Кроме того, для осуществления ряда действий доступны контекстные меню и перетаскивание с помощью мыши (рис. 2.34).

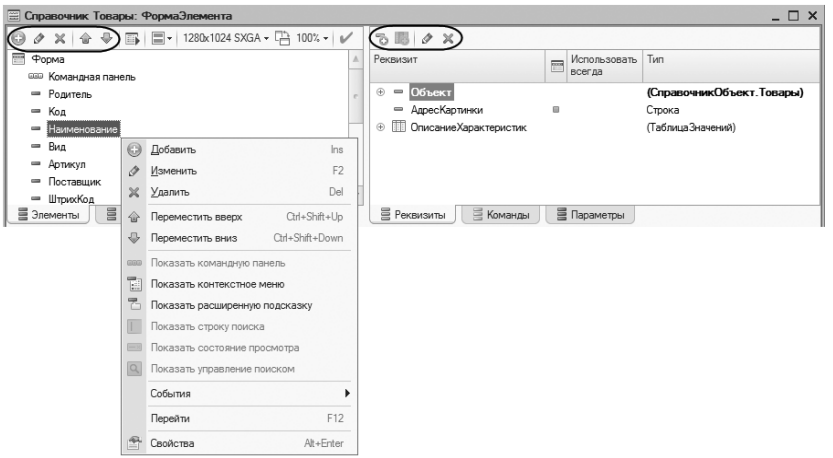


Рис. 2.34. Команды редактора формы

Помимо этого в командной панели окна элементов формы расположены команды, с помощью которых уже на этапе конструирования формы можно оценить ее внешний вид в интерфейсе приложения в различных режимах отображения, вариантах интерфейса, масштабах и т. д.

С помощью кнопки Вариант интерфейса можно выбрать нужный вариант интерфейса и в окне предварительного просмотра посмотреть, как выглядит форма в интерфейсе Такси или в интерфейсе Версия 8.2 (рис. 2.35).

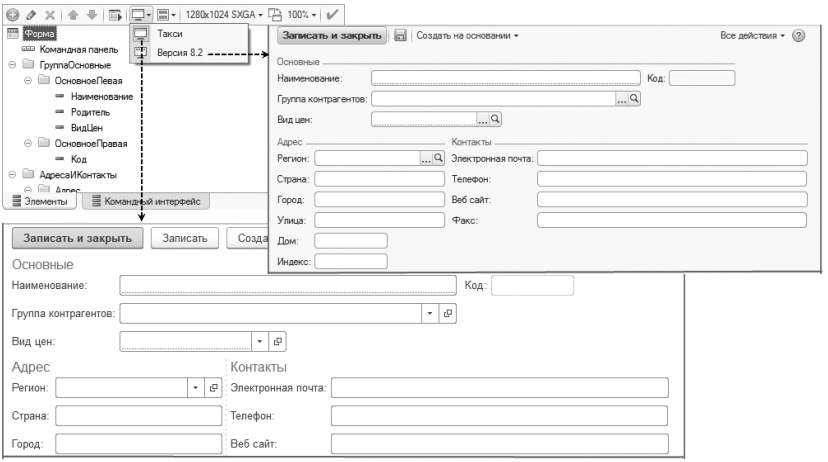


Рис. 2.35. Вид формы в редакторе в интерфейсах «Такси» и «Версия 8.2»

ПРИМЕЧАНИЕ

Кнопка Вариант интерфейса доступна только в том случае, когда свойство Режим совместимости интерфейса установлено в значение Такси. Разрешить Версия 8.2 или Версия 8.2. Разрешить Такси.

С помощью кнопки Отображение можно оценить, как выглядит форма в обычном и компактном режиме (рис. 2.36).

Рис. 2.36. Вид формы в редакторе в режимах отображения «Обычное» и «Компактное»

ПРИМЕЧАНИЕ

Кнопка Отображение доступна только в том случае, когда свойство Режим совместимости интерфейса установлено в значение Такси, Такси. Разрешить Версия 8.2 или Версия 8.2. Разрешить Такси и если у самой формы свойство Вариант масштаба установлено в значение Авто.

Если форма будет использоваться только в интерфейсе Такси, то можно изменять масштаб формы с помощью свойства Масштаб. Перед этим с помощью кнопки Масштаб в редакторе формы можно проверить, как форма будет отображаться с нестандартным масштабом (рис. 2.37).

ПРИМЕЧАНИЕ

Кнопка Масштаб доступна только в том случае, когда свойство Режим совместимости интерфейса установлено в значение Такси и если не используется компактный режим отображения формы.

О свойствах формы Масштаб и Вариант масштаба более подробно рассказывается в разделе «Вариант масштаба», «Масштаб» на стр. 275.

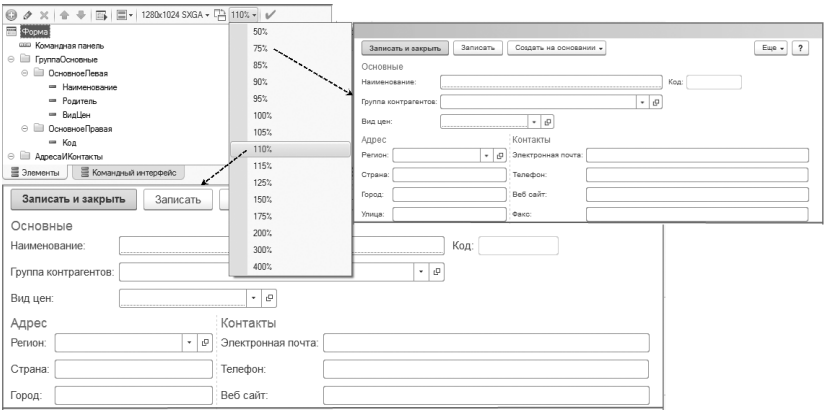


Рис. 2.37. Вид формы в редакторе в различных масштабах отображения

Поскольку устройства, на которых работают пользователи, имеют различные разрешения, то полезно проверить, как будет выглядеть форма при различных разрешениях дисплея. Это можно сделать с помощью кнопки Дисплей в командной панели редактора формы (рис. 2.38).

Поскольку мобильные устройства в процессе работы могут изменять ориентацию своего экрана, то полезно проверить, как будет выглядеть форма при смене ориентации. Это можно сделать с помощью кнопки Повернуть экран в командной панели редактора формы (рис. 2.39).

С помощью кнопки Проверить можно открыть форму не в окне предварительного просмотра, а в отдельном окне и посмотреть, как она будет выглядеть в интерфейсе приложения. После этого можно максимизировать окно формы или изменить размеры окна, как требуется (рис. 2.40).

Рис. 2.38. Выбор варианта разрешения дисплея для отображения формы в редакторе

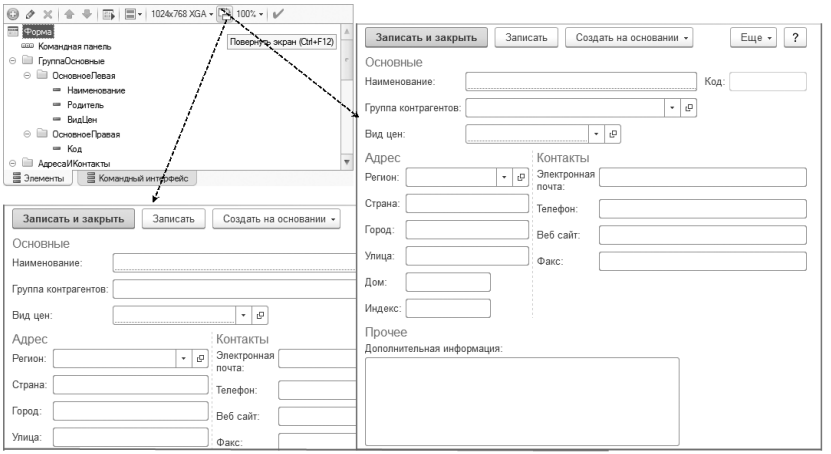


Рис. 2.39. Вид формы в редакторе при ландшафтной и портретной ориентации экрана

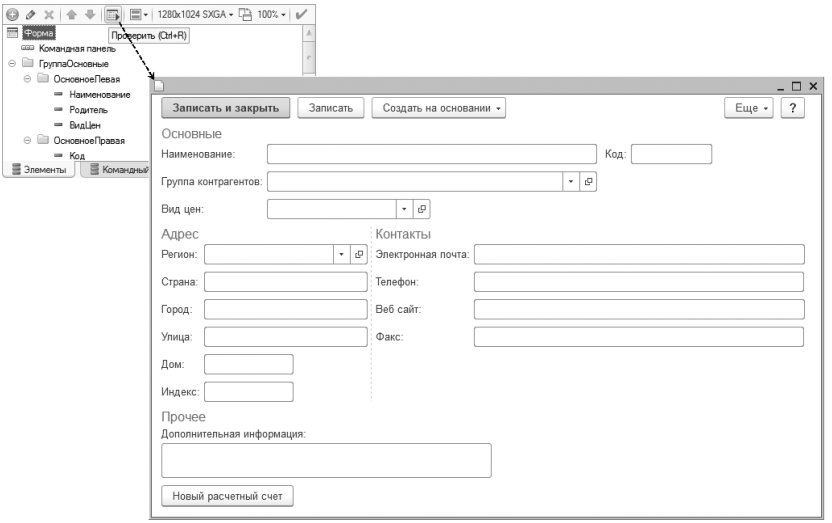


Рис. 2.40. Вид формы в интерфейсе приложения

Иерархия элементов формы

Форма описывается разработчиком прикладного решения в виде иерархического дерева на закладке Элементы редактора формы (рис. 2.41).

Все элементы формы подчинены корневому элементу Форма. Свойства элементов дерева зависят от свойств, которыми наделены элементы вышестоящего уровня. При этом каждый из элементов обладает своими уникальными свойствами, характерными только для него.

Влияние свойств элементов дерева друг на друга (имеются в виду одинаковые свойства, например Ширина) является обоюдным. Таким образом, свойства нижнего уровня могут оказывать влияние на свойства верхнего уровня иерархии, и наоборот. Примеры влияния свойств элементов дерева друг на друга рассматриваются в главе 2.9 на стр. 381.

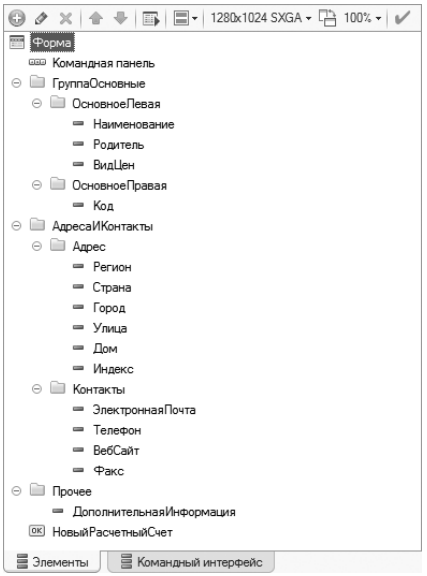


Рис. 2.41. Иерархия элементов формы

Свойства элементов верхнего уровня иерархии оказывают влияние на подчиненные элементы не только в процессе работы приложения, но и на этапе разработки. Такая зависимость достаточно хорошо видна при добавлении новых элементов формы. Так, например, свойство Вид элемента формы зависит от расположения элемента в иерархии элементов формы.

Свойства формы

Форма обладает рядом свойств, которые присущи ей всегда. Кроме того, ряд свойств формы определяется ее основным реквизитом. Именно основной реквизит формы определяет ее поведение, вид, состав команд, другие дополнительные возможности, предоставляемые разработчику формы и ее пользователю.

Для редактирования свойств формы необходимо воспользоваться контекстным меню (пункт меню Свойства) ветви Форма в окне элементов формы или просто дважды щелкнуть мышью на корне дерева элементов (рис. 2.42).

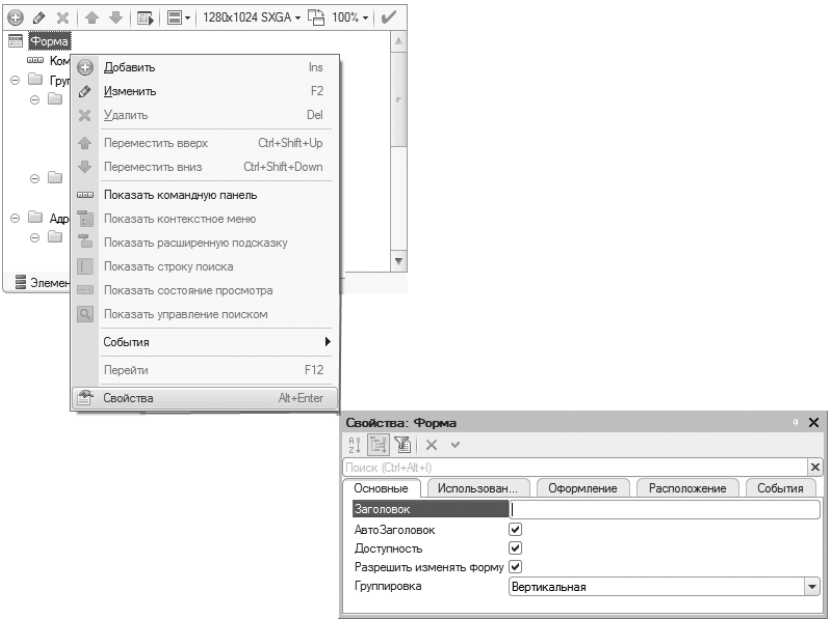


Рис. 2.42. Открытие окна «Свойства» формы

Палитра свойств формы содержит множество свойств, собранных в группы Основные, Использование, Оформление и т. п. Рассмотрим наиболее важные из этих свойств.

«Заголовок», «Автоматический заголовок», «Отображать заголовок»

Свойство Заголовок определяет заголовок формы, как его будут видеть разработчик в окне предварительного просмотра формы и пользователь во время работы с формой. Заголовок формы может быть дополнен системной информацией, полученной из свойств объекта конфигурации, связанных с расширенным представлением объектов, списков. Такое влияние на свойство Заголовок возможно, если установлено свойство АвтоЗаголовок, и рассматривается в главе 2.4 на стр. 327.

Например, если задать заголовок формы элемента справочника Товары – «Номенклатура», то при открытии товара в режиме

1С:Предприятие заголовок формы (рис. 2.43) будет состоять из строки «Номенклатура» плюс название конкретного товара («Босоножки»), плюс в скобках представление объекта конфигурации («Товар»).

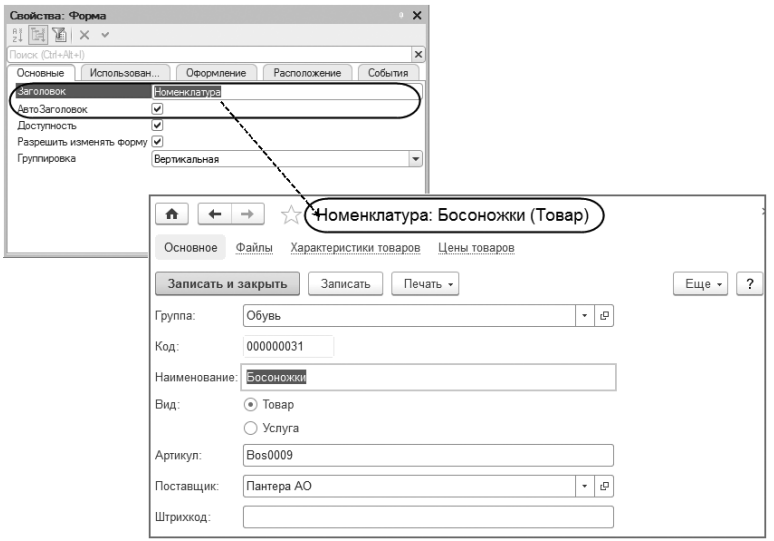


Рис. 2.43. Заголовок формы

Если свойство Заголовок не заполнено, а свойство АвтоЗаголовок включено, то заголовок формы товара будет сформирован полностью автоматически – «Босоножки (Товар)». При снятом автозаголовке и пустом заголовке заголовок формы будет отсутствовать.

С помощью свойства ОтображатьЗаголовок можно управлять показом заголовка формы (включая автозаголовок и признак модификации). Например, сняв флажок ОтображатьЗаголовок, можно сэкономить рабочее пространство экрана. Это особенно актуально для специализированных режимов основного окна приложения Рабочее место, Полноэкранное рабочее место и Киоск.

В режиме основного окна приложения Обычный свойство ОтображатьЗаголовок применяется только для форм, расположенных на начальной странице.

Подробнее о режимах основного окна приложения рассказывается в первой части книги, в разделе «Режимы основного окна приложения» на стр. 26.

Когда на начальной странице расположено несколько форм, то вверху всегда выводится надпись «Начальная страница», а над каждой формой – ее собственный заголовок. Можно при желании отключить показ заголовков форм на начальной странице (рис. 2.44).

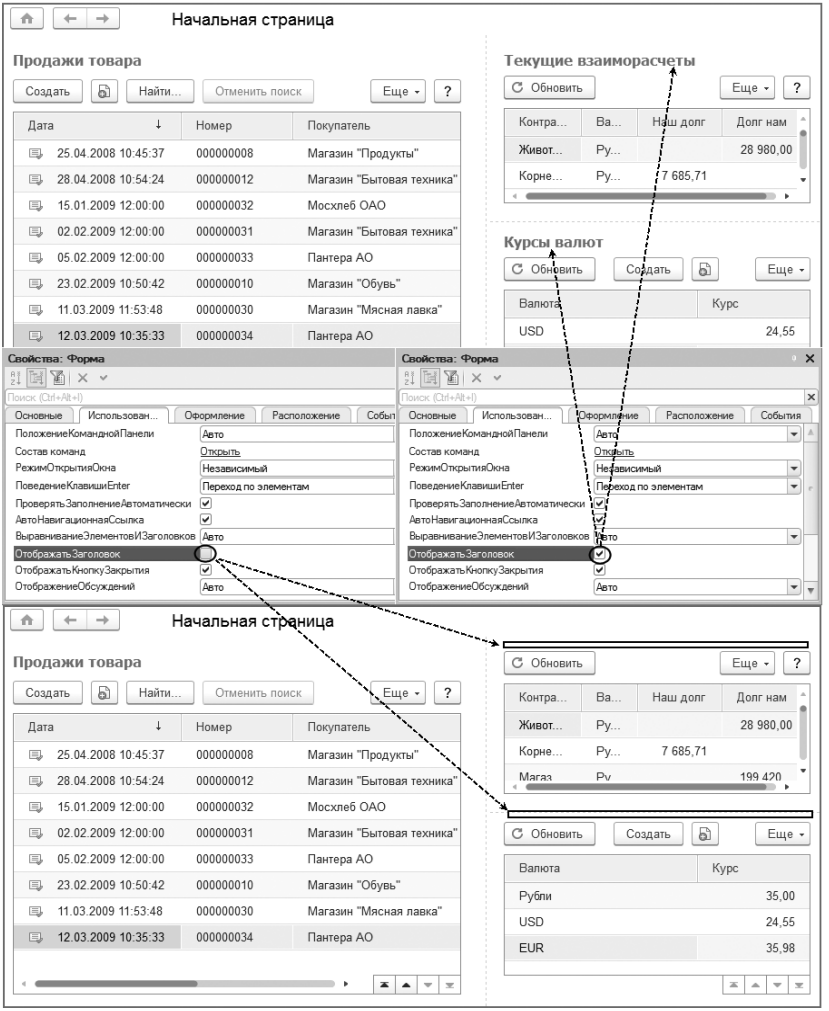


Рис. 2.44. Заголовки форм на начальной странице

Если же на начальной странице располагается только одна форма и свойство ОтображатьЗаголовок включено, то заголовок формы выводится вместо надписи «Начальная страница». Если флажок ОтображатьЗаголовок снят, то вместо заголовка формы выводится надпись «Начальная страница» (рис. 2.45).

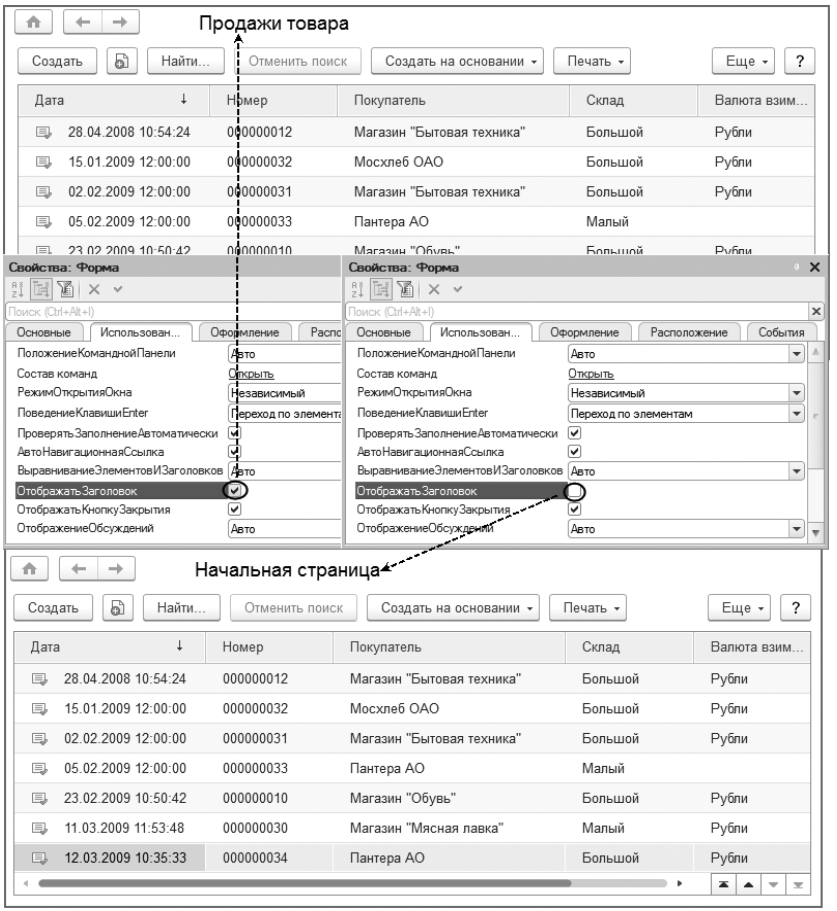


Рис. 2.45. Заголовок формы на начальной странице

Свойство ОтображатьЗаголовок применяется только в режиме Такси.

«Положение командной панели»

Свойство ПоложениеКоманднойПанели определяет положение командной панели формы. Свойство может принимать одно из четырех значений: Нет, Авто, Верх, Низ (рис. 2.46). По умолчанию системой устанавливается значение Авто. Это означает, что решение о расположении командной панели принимается платформой.

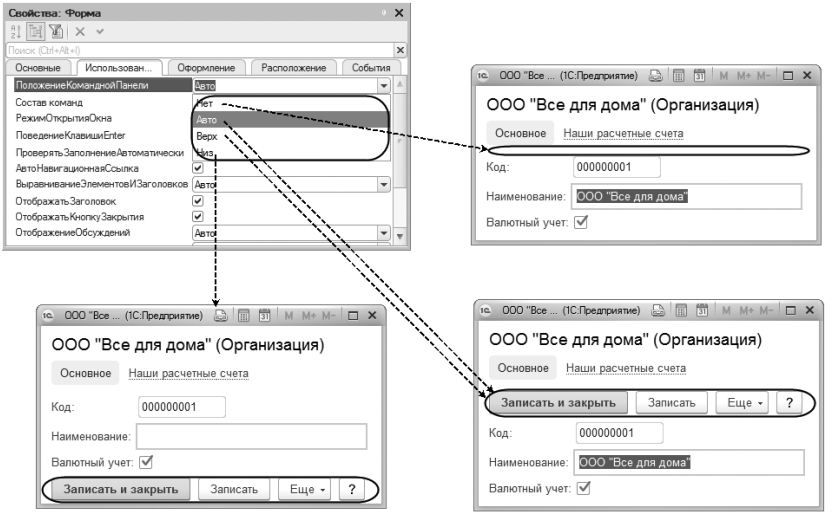


Рис. 2.46. Расположение командной панели формы

«Доступность»

Снятие флажка у свойства Доступность означает, что все элементы формы недоступны для редактирования пользователем. Кроме того, ограничить доступность формы можно с помощью свойства ТолькоПросмотр, которое доступно только из встроенного языка.

«Режим открытия окна»

Свойство РежимОткрытияОкна определяет, в каком окне будет открываться форма. Возможны значения Независимый, Блокировать окно владельца и Блокировать весь интерфейс.

В режиме Независимый форма открывается в рабочей области основного окна приложения. При этом окно формы не препятствует возможности переключения между окнами приложения. В таком режиме форма живет своей, не мешающей остальным формам жизнью.

В режиме Блокировать окно владельца форма открывается в блокирующем окне, при этом работа с формой, из которой инициировано открытие текущей формы, блокируется. Данный режим предназначен для форм, в которых вводится мало информации и работа с которыми не требует длительного времени, например для ввода элементов справочников, содержащих небольшое количество реквизитов. Этот режим внешне аналогичен модальному открытию формы, однако при открытии из встроенного языка работа модуля не останавливается на время работы открываемой формы. При открытии происходит блокировка только «родительского» окна, другие окна приложения остаются доступными.

В режиме Блокировать весь интерфейс форма открывается в блокирующем окне, при этом блокируется работа не только с родительской формой, но со всем интерфейсом прикладного решения. Окна, блокирующие весь интерфейс, обеспечивают работу приложения без использования модальных окон. Отказ от модальности является необходимым условием для работы «1С:Предприятия» в веб-клиенте и на мобильных устройствах, а также по ряду других причин. Подробнее об этом будет рассказано в разделе «Открытие формы в блокирующем режиме без использования модальности» на стр. 461.

«Проверять заполнение автоматически»

Установленный флажок у свойства ПроверятьЗаполнениеАвтоматически означает автоматическую проверку заполнения реквизитов формы при работе пользователя.

Механизм проверки заполнения реквизитов формы рассматривается в главе 2.8 на стр. 377 и в главе 3.11 на стр. 525.

«Условное оформление»

Гиперссылка Открыть у свойства УсловноеОформление позволяет открыть окно Настройка условного оформления. В данном окне можно настроить, например, выделение цветом некоторого элемента в зависимости от значений других элементов формы.

Примеры рассматриваются в разделе «Как настроить условное оформление динамического списка» на стр. 410 и в главе 3.15 на стр. 584.

В случае если требуется настроить условное оформление динамического списка, рекомендуется использовать для этого настройки самого списка, а не условное оформление формы.

«Разрешить изменять форму»

Свойство РазрешитьИзменятьФорму определяет возможность изменения формы в пользовательском режиме работы.

ВНИМАНИЕ!

Настройка форм в пользовательском режиме работы является стандартной возможностью, предоставляемой платформой. Это означает, что пользователь может изменить порядок элементов формы, перегруппировать их, изменить их видимость. С помощью свойства РазрешитьИзменятьФорму такую возможность можно пресечь.

Вместо запрещения изменения всей формы разработчик может применять «точечные» запреты для конкретных элементов формы. Свойства, с помощью которых можно запрещать изменения элементов формы, рассматриваются в разделах данной главы, посвященных элементам формы. Примеры настройки свойств рассматриваются в разделе «Как и зачем объединять элементы формы в группы» на стр. 381.

«Группировка»

Свойство Группировка определяет режим группировки подчиненных элементов формы. Возможные значения свойства: Вертикальная, Горизонтальная и Горизонтальная если возможно.

При значении Вертикальная элементы формы располагаются сверху вниз. При значении Горизонтальная элементы будут располагаться

слева направо. При значении Горизонтальная если возможно применяется горизонтальная группировка при наличии достаточного места по ширине формы. В противном случае – вертикальная группировка (рис. 2.47).

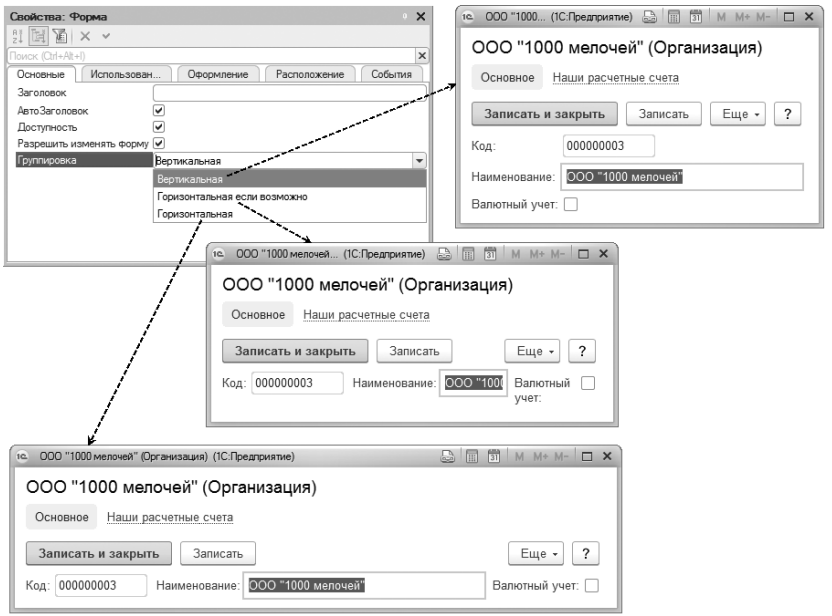


Рис. 2.47. Режимы группировки элементов формы

Настройка влияет только на элементы верхнего уровня иерархии подчиненных элементов. Порядок расположения элементов формы, находящихся внутри групп, определяется свойством Группировка соответствующей группы элементов.

«Вертикальная прокрутка»

С помощью свойства ВертикальнаяПрокрутка можно управлять отображением вертикальной прокрутки формы, а также тем, как будут вести себя элементы формы в том случае, если форма будет сжиматься по вертикали.

Возможны следующие режимы включения вертикальной прокрутки формы:

- Использовать – полоса вертикальной прокрутки появится после того, как вертикальный размер элементов формы достигнет размера, который был задан при редактировании формы в конфигураторе;
- Использовать без растягивания – при растягивании формы элементы формы не будут реагировать на изменение размера формы;
- Использовать при необходимости – перед появлением полосы прокрутки элементы формы будут сжиматься по вертикали до своего минимального размера. Если они все-таки не поместятся в форме, появится полоса прокрутки;
- Авто – поведение формы зависит от типа основного реквизита формы. Если основной реквизит формы имеет тип ДинамическийСписок или ОтчетОбъект, то свойство ВертикальнаяПрокруткапринимаетзначениеИспользоватьпринеобходимости. В остальных случаях – значение Использовать.

На рис. 2.48 продемонстрированы различия режимов включения вертикальной прокрутки формы.

При значении Использовать (см. рис. 2.48 слева внизу) вертикальная прокрутка появляется сразу же, как только форма сжимается по вертикали относительно своего исходного размера. При этом все элементы формы, в т. ч. картинка товара, имеют свой реальный размер.

При значении Использовать при необходимости элементы формы сжимаются так, чтобы избежать вертикальной прокрутки формы. В частности, при такой же высоте формы, что и в предыдущем случае, картинка товара сжимается и полоса прокрутки отсутствует (см. рис. 2.48 справа внизу). Но если элементы формы сжаты максимально и все равно не умещаются по высоте формы, то вертикальная прокрутка формы появляется (см. рис. 2.48 справа вверху).

Стандартно свойство ВертикальнаяПрокрутка устанавливается в значение Авто. Менять его, как правило, нет необходимости. Исключением являются формы, имеющие поля для вывода HTML-документов, форматированных документов, текстовых документов,

табличных документов (кроме форм отчетов). Для таких форм свойство Вертикальная прокрутка должно быть установлено в значение Использовать при необходимости.

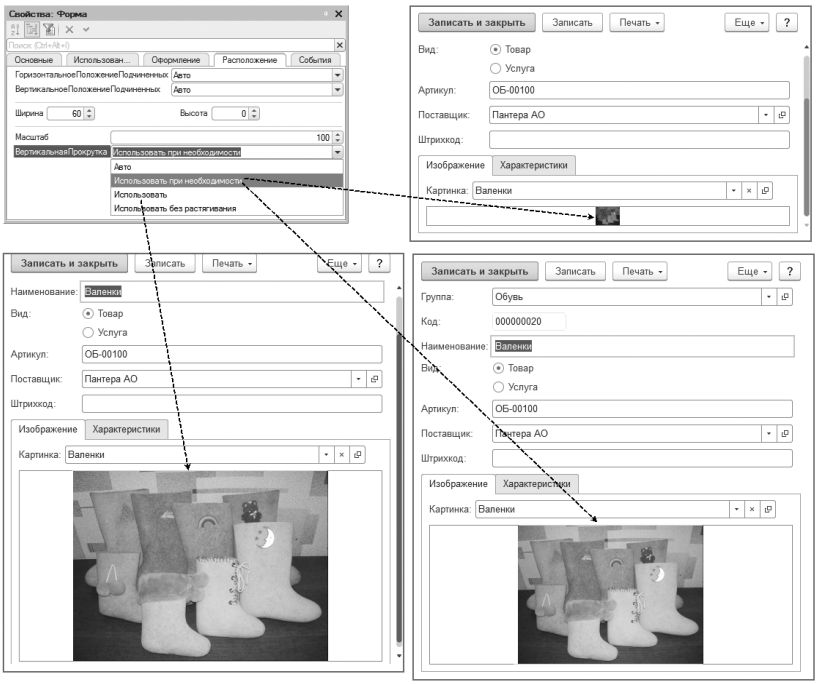


Рис. 2.48. Режимы вертикальной прокрутки формы

«Вариант масштаба», «Масштаб»

Свойство ВариантМасштаба определяет, каким образом форма будет отображаться в интерфейсе Такси:

- Авто – режим отображения формы определяется автоматически на основе анализа значения свойства `ВариантМасштабаФормКлиентскогоПриложения` объекта `НастройкиКлиентскогоПриложения`. Если в данном свойстве не указано конкретное значение варианта масштаба, то в интерфейсе Такси значение Авто будет трактоваться как Обычный.

- Обычный – форма выглядит стандартно для интерфейса Такси. На размер формы влияет свойство Масштаб.
- Компактный – в этом случае форма выглядит «почти» как в интерфейсе версии 8.2. Свойство Масштаб не оказывает влияния на размер формы.

С помощью свойства ВариантМасштаба можно управлять отображением одной формы, разрабатываемой в конфигураторе. Она может выглядеть в обычном или компактном масштабе.

Чтобы изменить вариант масштаба сразу всех форм прикладного решения, нужно программно установить значение свойства ВариантМасштабаФормКлиентскогоПриложения объекта НастройкиКлиентскогоПриложения.

Компактный вариант масштаба может использоваться для быстрого перевода сложных насыщенных форм с интерфейса Версия 8.2 на интерфейс Такси. Чтобы, не переделывая саму форму, быстро уменьшить масштаб ее отображения примерно до 80 %.

Но если форма будет использоваться только в интерфейсе Такси, то вместо этого лучше уменьшать/увеличивать масштаб формы с помощью свойства Масштаб. Данное свойство указывает, каким должен быть масштаб данной формы относительно ее базового размера (в процентах). Значение свойства может изменяться от 10 до 400. Свойство Масштаб не применяется для форм с компактным изображением.

Масштаб, установленный в клиентском приложении, накладывается на масштаб, указанный в конфигураторе для каждой формы. Таким образом, если в конфигураторе установлен масштаб формы 70 %, а в клиентском приложении установлен масштаб 50 %, то результирующий масштаб будет 35 % от реального размера формы.

«Список групп»

С помощью свойства СписокГрупп можно поддерживать автоматическую синхронизацию данных в форме, когда в ней отображаются одновременно список групп и собственно содержимое иерархического справочника. Это свойство становится доступным в том случае, если основным реквизитом формы выступает реквизит типа ДинамическийСписок.

Для этого в свойстве формы СписокГрупп нужно указать таблицу, отображающую динамический список, который показывает только иерархию групп отображаемого справочника (рис. 2.49).

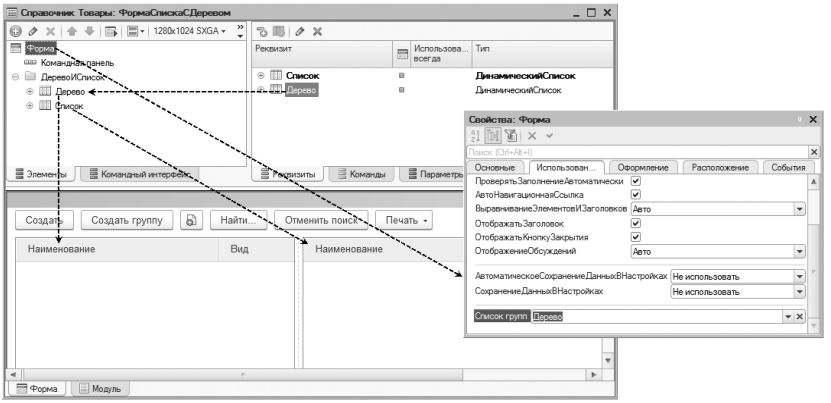


Рис. 2.49. Установка свойства «Список групп» для формы, отображающей данные двух связанных списков

В этом случае при выборе группы в списке групп в списке элементов будет отображаться содержимое выбранной группы, и наоборот (рис. 2.50).

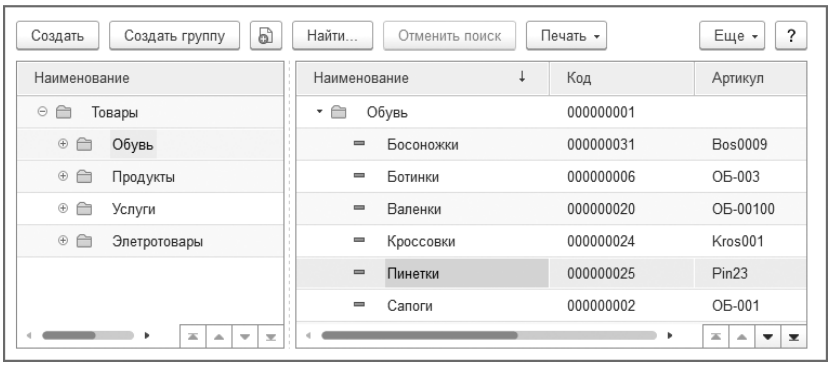


Рис. 2.50. Автоматическая синхронизация данных в списке групп и в списке элементов иерархического справочника

Виды элементов формы

В форме используются следующие элементы:

- Группа:
  - Обычная группа;
  - Группа – Страницы;
  - Группа – Командная панель;
  - Группа колонок.
- Поле.
- Кнопка.
- Таблица.
- Декорация:
  - Декорация – Надпись;
  - Декорация – Картинка.
- Дополнение элемента формы.

Для того чтобы добавить в состав формы новый элемент определенного вида, необходимо воспользоваться одним из этих элементов. Для отображения различных данных прикладного решения необходимо изменять свойство элемента формы Вид.

При добавлении нового элемента редактором формы будет проанализировано свойство Данные (с каким реквизитом будет связан элемент формы). В зависимости от этого платформа позволит выбрать один из нескольких возможных видов этого элемента.

Открытие свойств элементов формы ничем не отличается от рассматривавшегося ранее открытия свойств формы.

Поле

Элемент формы Поле предназначен для отображения примитивных типов данных, текстовых, табличных, HTML-документов, диаграмм, календарей, индикаторов и др. Тип данных, которые отображает элемент Поле, влияет на то, какие значения может принимать его свойство Вид.

- Поле надписи. Поле, недоступное для непосредственного редактирования в пользовательском режиме работы (рис. 2.51). Надпись

изменяется из встроенного языка. Для отображения надписи в форме нужно создать реквизит типа Строка и связать его с полем формы вида Поле надписи.

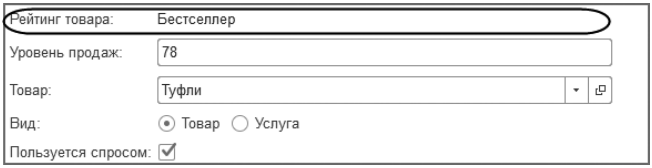


Рис. 2.51. Элемент формы «Поле надписи»

- Поле ввода. Поле, допускающее редактирование данных. Это могут быть как примитивные типы данных (например, число), так и ссылочные данные (например, ссылки на элементы справочника). Для отображения поля ввода в форме нужно создать реквизит требуемого типа и связать его с полем формы вида Поле ввода (рис. 2.52).

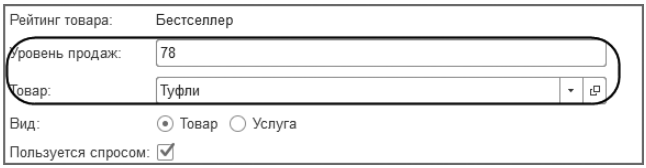


Рис. 2.52. Элемент формы «Поле ввода»

- Поле переключателя. Поле, позволяющее выбрать один из нескольких вариантов значения с помощью отметки нужного (рис. 2.53). Для отображения переключателя в форме нужно создать реквизит типа Число или Строка, связать его с полем формы вида Поле переключателя и задать у этого поля список значений для выбора в свойстве СписокВыбора.

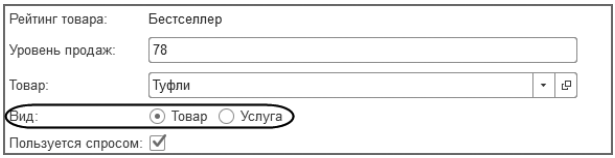


Рис. 2.53. Элемент формы «Поле переключателя»

- Поле флажка. Поле, предназначенное для отображения или установки одного из значений (рис. 2.54). Для отображения флажка в форме нужно создать реквизит типа Булево или Число и связать его с полем формы вида Поле флажка.

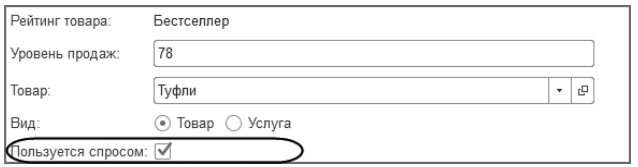


Рис. 2.54. Элемент формы «Поле флажка»

- Поле индикатора. Поле, предназначенное для графического отображения текущего состояния реквизита формы (рис. 2.55). Для отображения индикатора в форме нужно создать реквизит типа Число и связать его с полем формы вида Поле индикатора.

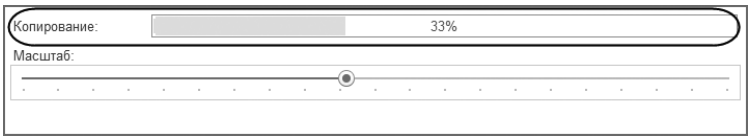


Рис. 2.55. Элемент формы «Поле индикатора»

- Поле полосы регулирования. Поле, предназначенное для ввода числовых данных с помощью шкалы (рис. 2.56). Для отображения полосы регулирования в форме нужно создать реквизит типа Число и связать его с полем формы вида Поле полосы регулирования.

Рис. 2.56. Элемент формы «Поле полосы регулирования»

- Поле картинки. Поле, отображающее картинку. Для отображения картинки в форме нужно создать реквизит типа Картинка

и связать его с полем вида Поле картинки. Например, таким образом на рис. 2.57 (справа) показана картинка из библиотеки картинок. Или же, если картинка хранится в реквизите объекта, можно прочитать навигационную ссылку на картинку в строковый реквизит формы и связать поле формы вида Поле картинки с этим реквизитом (рис. 2.57 слева).

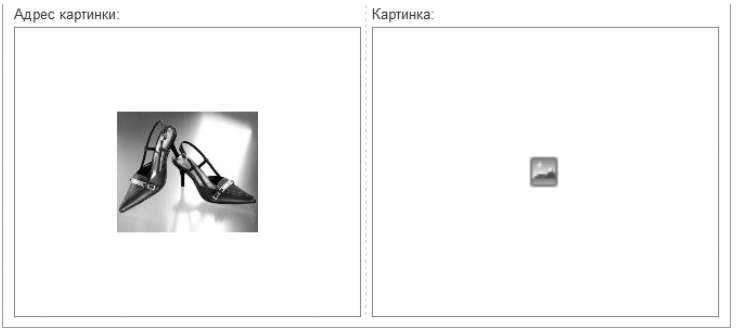


Рис. 2.57. Элемент формы «Поле картинки»

- Поле календаря. Поле, предназначенное для выбора и отображения даты в виде календаря (рис. 2.58). Для отображения календаря в форме нужно создать реквизит типа `Дата` и связать его с полем вида Поле календаря.

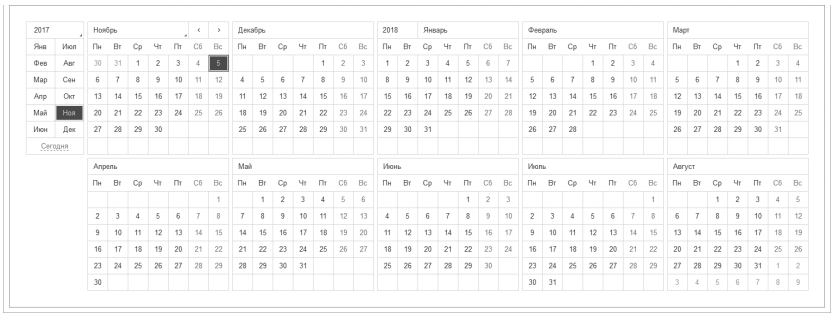


Рис. 2.58. Элемент формы «Поле календаря»

- Поле текстового документа. Поле для редактирования и просмотра текстовых данных (рис. 2.59). Для отображения текстового документа в форме нужно создать реквизит типа

ТекстовыйДокумент или Строка и связать его с полем вида Поле текстового документа.

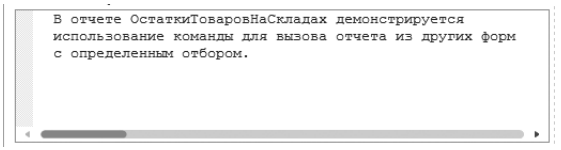


Рис. 2.59. Элемент формы «Поле текстового документа»

- Поле табличного документа. Поле табличного вида (совокупность строк и столбцов), в котором отображаются и редактируются данные. Как правило, используется для отображения результатов работы отчетов и печатных форм. Может использоваться для ввода данных (рис. 2.60). Для отображения табличного документа в форме нужно создать реквизит типа ТабличныйДокумент и связать его с полем вида Поле табличного документа.

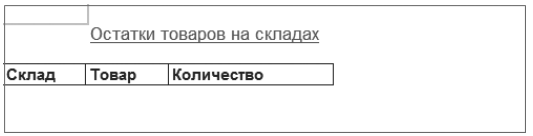


Рис. 2.60. Элемент формы «Поле табличного документа»

- Поле форматированного документа. Поле для редактирования и просмотра текстовых данных различных форматов с возможностью их оформления (рис. 2.61). Для отображения форматированного документа в форме нужно создать реквизит типа ФорматированныйДокумент и связать его с полем вида Поле форматированного документа.

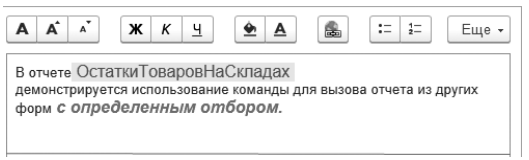


Рис. 2.61. Элемент формы «Поле форматированного документа»

- Поле HTML-документа. Поле для редактирования и просмотра HTML-ресурсов (рис. 2.62). Для отображения HTML-документа в форме нужно создать реквизит типа `Строка` и связать его с полем вида Поле HTML-документа. Строковой реквизит может содержать адрес для доступа к веб-контенту (URL интернет-ресурса) или сам HTML-код.

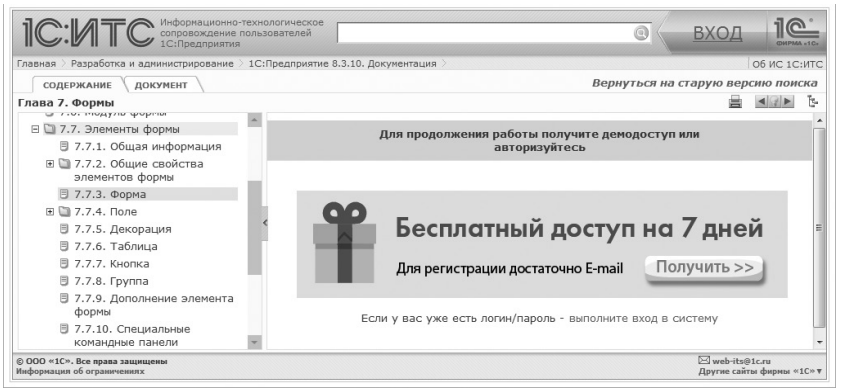


Рис. 2.62. Элемент формы «Поле HTML-документа»

- Поле диаграммы. Поле, отображающее различные виды диаграмм, представляющих данные в графическом виде (рис. 2.63). Для отображения диаграммы в форме нужно создать реквизит типа `Диаграмма` и связать его с полем вида Поле диаграммы.

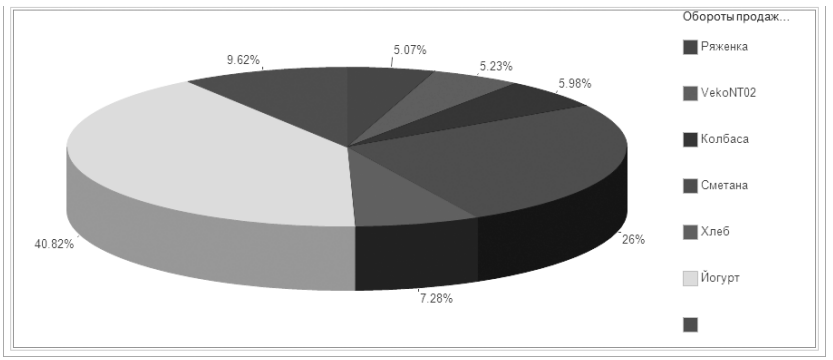


Рис. 2.63. Элемент формы «Поле диаграммы»

- Поле диаграммы Ганта. Поле, отображающее диаграмму Ганта. Диаграмма Ганта является способом отражения длительности и последовательности процессов, которые показываются в виде полос, расположенных вдоль оси времени (рис. 2.64). Для отображения диаграммы Ганта в форме нужно создать реквизит типа `ДиаграммаГанта` и связать его с полем вида Поле диаграммы Ганта.

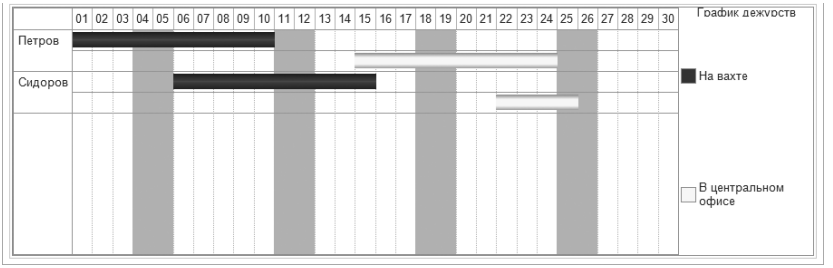


Рис. 2.64. Элемент формы «Поле диаграммы Ганта»

- Поле дендрограммы. Поле, отображающее дендрограмму связей объектов. Дендрограмма представляет собой графический способ отображения степени сходства объектов (рис. 2.65). Для отображения дендрограммы в форме нужно создать реквизит типа `Дендрограмма` и связать его с полем вида Поле дендрограммы.

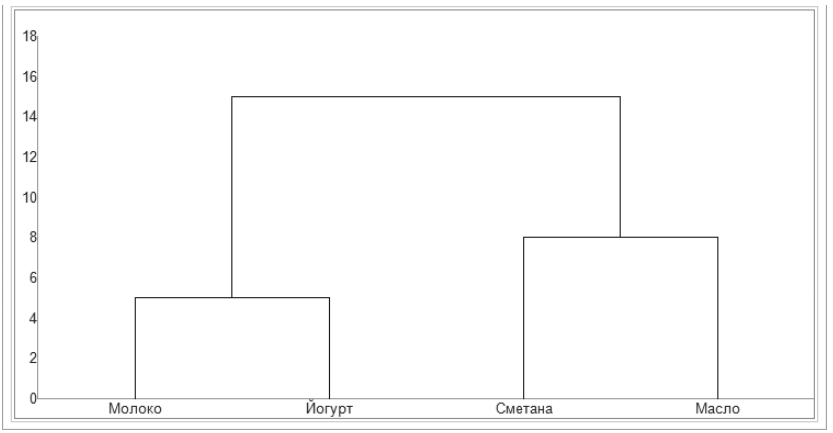


Рис. 2.65. Элемент формы «Поле дендрограммы»

- Поле графической схемы. Поле для просмотра и редактирования графических схем, созданных в специализированном редакторе, входящем в состав «1С:Предприятия» (рис. 2.66). Для отображения графической схемы в форме нужно создать реквизит типа `ГрафическаяСхема` и связать его с полем вида Поле графической схемы.

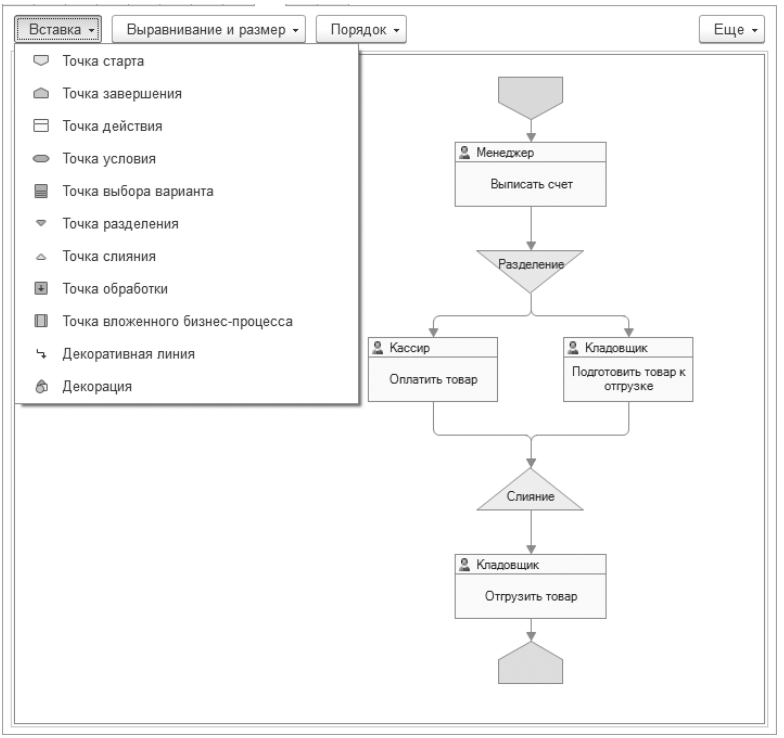


Рис. 2.66. Элемент формы «Поле графической схемы»

- Поле географической схемы. Поле для просмотра итоговых данных в разрезе их географического положения (рис. 2.67). Для отображения географической схемы в форме нужно создать реквизит типа `ГеографическаяСхема` и связать его с полем вида Поле географической схемы.

Рис. 2.67. Элемент формы «Поле географической схемы»

- Поле планировщика. Поле для просмотра и редактирования запланированных событий, привязанных к дате и времени, с возможностью повторения этих событий с определенной периодичностью (рис. 2.68). Для отображения планировщика в форме нужно создать реквизит типа `Планировщик` и связать его с полем вида Поле планировщика.

Расписание занятий:

	понедельник, 6 ноября 2		вторник, 7 ноября 2017		среда, 8 ноября 2017		четверг, 9 ноября 2017		пятница, 10 ноября 2017
08:00		08:00	русский язык	08:00		08:00	геометрия	08:00	физика
09:00		09:00	литература	09:00	алгебра	09:00	литература	09:00	
10:00		10:00	физкультура	10:00	химия	10:00	литература	10:00	астрономия
11:00		11:00	физика	11:00		11:00	французкий язык	11:00	история
12:00		12:00		12:00	биология	12:00		12:00	
13:00		13:00	история	13:00	география	13:00	классный час	13:00	французский язык
14:00		14:00		14:00	театр. студия	14:00	литературный кружок	14:00	химия

Рис. 2.68. Элемент формы «Поле планировщика»

- Поле периода. Поле для выбора периода дат, кратных месяцу (рис. 2.69). Для отображения поля периода в форме нужно создать реквизит типа `СтандартныйПериод` и связать его с полем вида Поле периода.

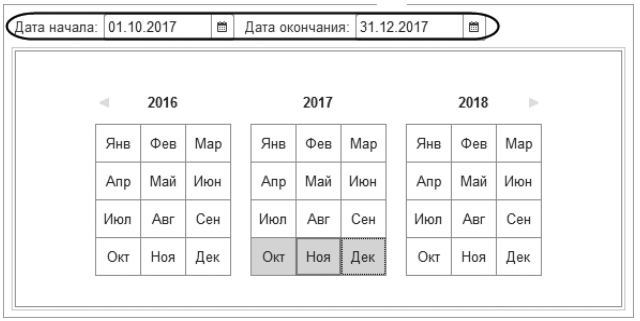


Рис. 2.69. Элемент формы «Поле периода»

Группа

Элемент формы Группа предназначен для группировки других элементов формы. Это могут быть группы полей, группы страниц, группы команд. Также для элементов типа Таблица можно создавать группы колонок. Группы могут быть вложенными друг в друга.

Свойство Вид элемента формы Группа может принимать значения:

- Обычная группа. Элемент формы, предназначенный для логической группировки других элементов формы (рис. 2.70). Элемент Обычная группа может не выделяться в форме, выполняя при этом функции группировки других элементов. Выделением группы управляет свойство Отображение.

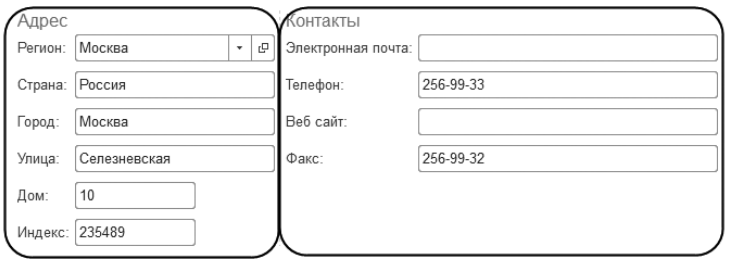


Рис. 2.70. Элементы формы «Обычная группа»

- Командная панель. Элемент формы, предназначенный для группировки кнопок и групп команд. Наполнение командной панели конкретными командами определяется свойством Источник

команд. На рис. 2.71 источником команд является таблица формы, отображающая динамический список.

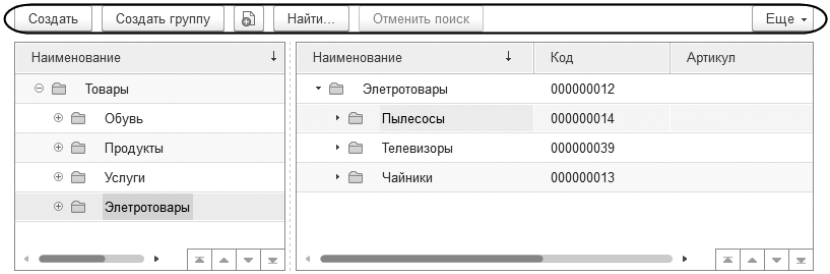


Рис. 2.71. Элемент формы «Командная панель»

Помимо отдельных кнопок в командную панель можно добавлять подчиненные группы следующего вида (рис. 2.72):

- Подменю. Элемент формы, представляющий собой выпадающее меню.
- Группа кнопок. Элемент формы, позволяющий сгруппировать кнопки в логические группы. Сам элемент не отображается на форме, но кнопки в группе можно расположить компактно (без отступов друг от друга).

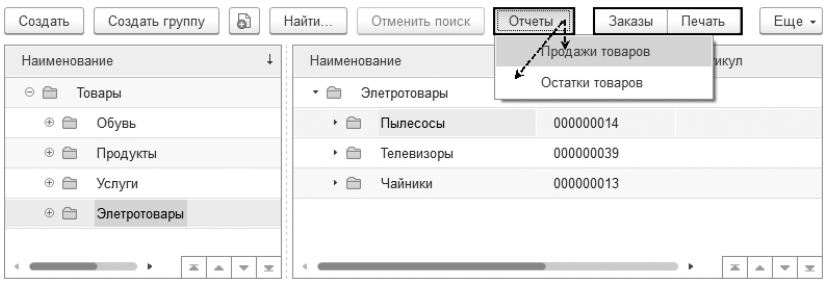


Рис. 2.72. Элементы командной панели «Подменю» и «Группа кнопок»

- Страницы. Элемент формы, предназначенный для организации панели с закладками. Для того чтобы добавить на такую панель страницы, необходимо добавить столько вложенных групп вида Страница, сколько страниц должно быть у панели (рис. 2.73).
  - Страница. Элемент формы, который может быть создан только как подчиненный элементу Страницы. Представляет собой подчиненную группу-страницу с данными (в нашем случае открыта страница «Изображение»). В пользовательском режиме работы страницы, на которых отсутствуют данные, не отображаются.

Рис. 2.73. Элемент формы «Страницы», состоящий из двух страниц: «Изображение» и «Характеристики»

- Группа колонок – позволяет объединять колонки в таблице (рис. 2.74). С помощью группы этого вида можно изменять правило группировки колонок.

Период	Регистратор	Номер строки	Контрагент	Сумма / Валюта
+ 11.03.2009 11:52:14	Поступление товара 000000044 от 11.03.2009 ...	1	Магазин "Продукты"	22 500,00
− 11.03.2009 11:53:48	Продажа 000000030 от 11.03.2009 11:53:48	1	Магазин "Мясная лавка"	48,86 Рубли
− 12.03.2009 10:35:33	Продажа 000000034 от 12.03.2009 10:35:33	1	Пантера АО	800,00 Рубли

Рис. 2.74. Элемент таблицы формы «Группа колонок»

Таблица

Элемент формы Таблица предназначен для отображения и редактирования различных табличных данных (рис. 2.75). Это может быть динамический список, табличная часть, список значений и т. д. У таблицы формы могут быть свои командные панели, контекстные меню. Поля колонок элемента Таблица могут быть сгруппированы (см. рис. 2.74).

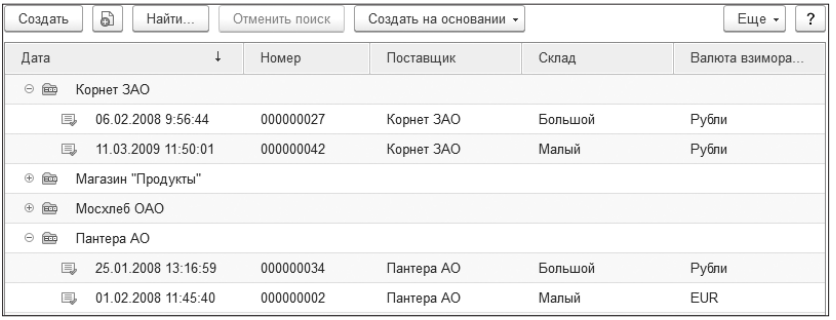


Рис. 2.75. Элемент формы «Таблица»

Кнопка

Элемент формы Кнопка предназначен для отображения кнопок и гиперссылок, при нажатии на которые выполняются связанные с ними команды. Кнопки могут быть подчинены командной панели. В зависимости от этого свойство Вид элемента формы Кнопка может принимать значения:

- Кнопка, Кнопка командной панели. Элемент представляет собой обыкновенную кнопку, расположенную непосредственно в форме или в составе командной панели (рис. 2.76). Если для кнопки не назначена команда, которая будет выполняться при нажатии, то кнопка в пользовательском режиме не отображается.
- Гиперссылка, Гиперссылка командной панели. Кнопка формы или кнопка командной панели, отображающаяся в виде гиперссылки (рис. 2.77). Если для гиперссылки не назначена команда, которая будет выполняться при нажатии, то гиперссылка в пользовательском режиме не отображается.

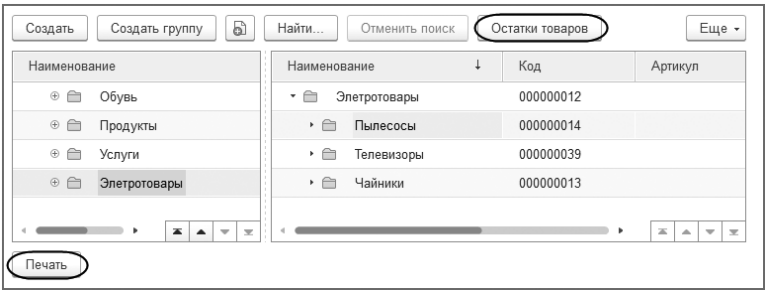


Рис. 2.76. Элементы формы «Кнопка» и «Кнопка командной панели»

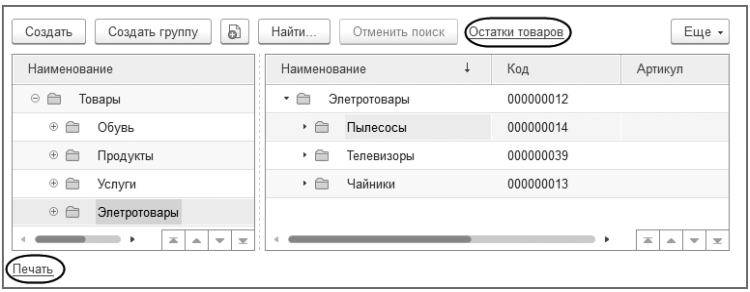


Рис. 2.77. Элементы формы «Гиперссылка» и «Гиперссылка командной панели»

Декорация

Элемент формы Декорация представляет собой оформительский элемент формы. Декорация может представлять собой надпись или картинку (рис. 2.78). Декорация не связана с данными (реквизитами формы) и обычно выводится для текстовых пояснений, не изменяемых из встроенного языка.

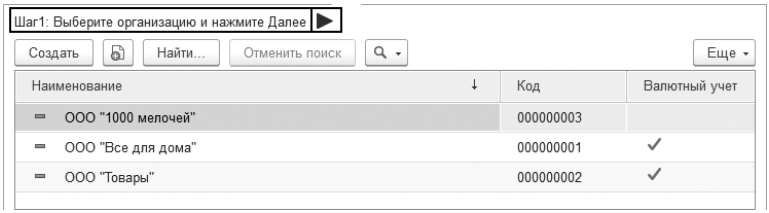


Рис. 2.78. Элемент формы «Декорация»

Дополнение элемента формы

Дополнение элемента формы предназначено для отображения дополнительных свойств элемента формы (на момент написания книги это дополнения таблицы, отображающей данные динамического списка) и управления этими элементами. Существуют следующие дополнения:

- Отображение строки поиска. Позволяет настроить отображение строки поиска.
- Состояние просмотра. Позволяет настроить отображение перечня примененных поисковых запросов. Не может располагаться в командной панели.
- Управление поиском. Позволяет настроить внешний вид кнопки управления поиском (рис. 2.79).

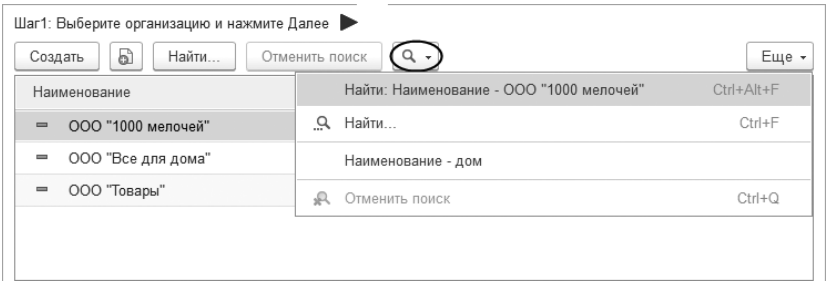


Рис. 2.79. Элемент формы «Дополнение элемента формы»

Контекстное меню элементов формы

Для всех элементов формы характерно наличие контекстного меню, вызываемого с помощью мыши или клавиатуры. В большинстве случаев во время работы приложения контекстные меню формируются платформой автоматически. Однако разработчик может вмешаться в этот процесс и повлиять на состав контекстного меню.

Чтобы получить доступ к контекстному меню элемента формы, нужно из контекстного меню элемента формы в редакторе формы выполнить команду Показать контекстное меню (рис. 2.80).

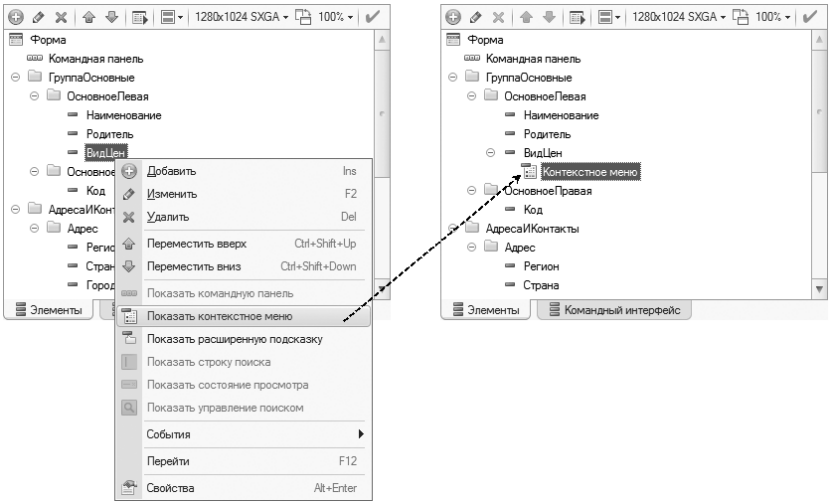


Рис. 2.80. Доступ к контекстному меню элемента формы

После этого появится возможность добавлять в контекстное меню элемента собственные команды.

Если разработчик желает модифицировать автоматически формируемое контекстное меню элемента формы, то флажок Автозаполнение в свойствах Контекстное меню элемента снимать не следует (рис. 2.81). В случае необходимости формирования полностью своего контекстного меню флажок следует снять.

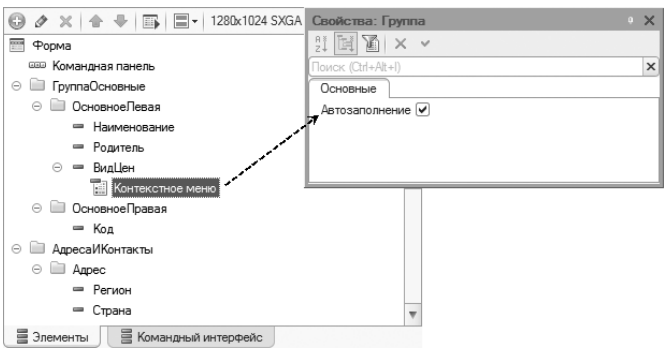


Рис. 2.81. Автозаполнение контекстного меню

Свойства элементов формы

Свойства элементов формы оказывают влияние как на внешний вид элементов в пользовательском режиме работы, так и на функциональность этих элементов. Перечень и описание всех свойств элементов формы не входят в задачи книги. В рамках текущей главы будут рассмотрены лишь некоторые из них. Свойства, связанные с проверкой заполнения элементов формы, рассматриваются в главе 2.9 на стр. 381. Свойства, оказывающие влияние на видимость элементов формы, рассматриваются в главе 2.8 на стр. 377.

Общие свойства

Свойство элемента формы Имя предназначено для задания уникального имени элемента в пределах формы. По имени можно обращаться к элементам формы из встроенного языка.

Свойство Заголовок позволяет системе сформировать понятную для пользователя подпись элемента формы. Для многих элементов формы доступно свойство ПоложениеЗаголовка со значениями: Нет, Авто, Лево, Верх, Право, Низ, которое позволяет управлять расположением подписи элемента формы.

С помощью свойства ПутьКДанным определяется связь элемента формы с теми данными, которые будут в нем отображаться и редактироваться. Если эта связь не задана, элемент не будет отображаться в форме.

Аналогичным образом, если для кнопки в свойстве ИмяКоманды не назначена команда, которая будет выполняться при ее нажатии, то кнопка в форме не будет отображаться.

При установленном свойстве ПропускатьПриВводе элемент будет пропускаться при обходе элементов формы клавишей Enter.

Активизировать элемент формы при ее открытии можно, установив свойство АктивизироватьПоУмолчанию. Это поможет избавить пользователя от лишних действий по переходу на нужный элемент формы и обратит его внимание на самый важный (с точки зрения работы приложения) элемент.

Элемент формы может находиться в режиме, запрещающем любые изменения, если в конфигураторе или с помощью встроенного языка у него (или у группы, в которую он входит) установлено свойство ТолькоПросмотр. Или же если реквизит формы, связанный с этим элементом, имеет признак Сохраняемые данные и у формы установлен режим ТолькоПросмотр.

Практически у всех элементов формы есть свойства АвтоМаксимальнаяШирина, АвтоМаксимальнаяВысота. Если эти свойства включены, то система автоматически рассчитывает максимальную ширину/высоту данного элемента.

Стандартно эти свойства включены, и в большинстве случаев не рекомендуется их отключать: это может негативно повлиять на размер и выравнивание элементов в форме. Также крайне не рекомендуется указывать фиксированную ширину и высоту элементов в свойствах Ширина, Высота (см. раздел «Как усовершенствовать внешний вид формы» на стр. 413).

У всех элементов формы есть свойства Подсказка, ОтображениеПодсказки и РасширеннаяПодсказка, с помощью которых можно пояснить пользователю назначение элемента. Во многих случаях наличие подсказок может избавить пользователя от чтения инструкций.

Краткую подсказку можно задать в свойстве Подсказка и сделать ее всплывающей при наведении указателя мыши на элемент. Для этого свойство ОтображениеПодсказки нужно установить в значение Всплывающая или Авто (рис. 2.82).

Кроме того, у элемента формы можно задать более подробную расширенную подсказку. Такая подсказка может быть оформлена в виде форматированного текста, включать картинки и ссылки.

Для получения доступа к расширенной подсказке элемента формы нужно из контекстного меню элемента в дереве элементов формы выполнить команду Показать расширенную подсказку (рис. 2.83).

Затем нужно задать заголовок расширенной подсказки, установив флажок Форматированная строка, и в свойствах самого элемента формы установить свойство ОтображениеПодсказки в значение Кнопка.

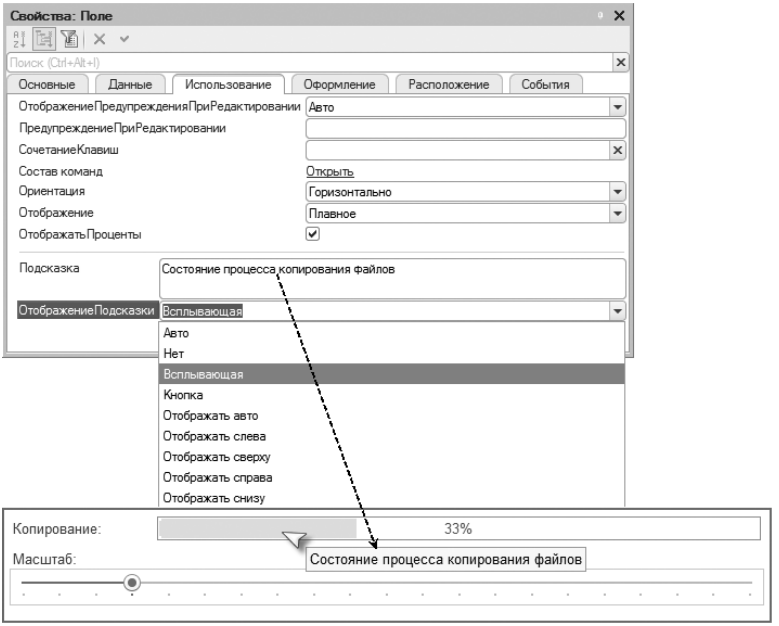


Рис. 2.82. Отображение подсказки у элемента формы

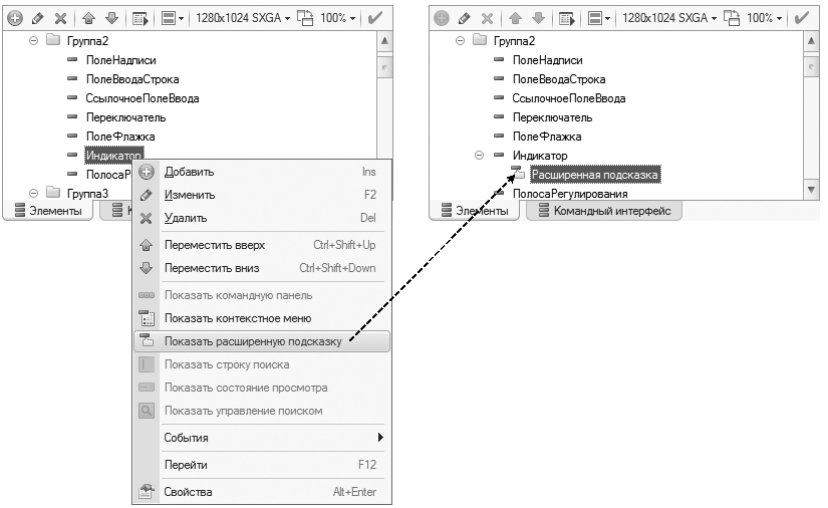


Рис. 2.83. Доступ к расширенной подсказке элемента формы

В результате в пользовательском режиме рядом с элементом формы появится кнопка в виде вопросительного знака, при нажатии на которую откроется подробная, красиво оформленная подсказка, поясняющая назначение этого элемента (рис. 2.84).

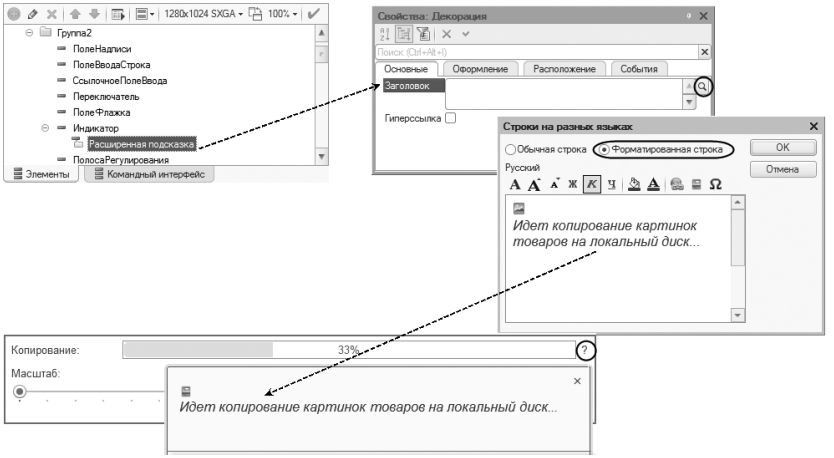


Рис. 2.84. Отображение расширенной подсказки у элемента формы

Если у элемента формы заданы и обычная, и расширенная подсказка, то при нажатии на кнопку вопроса будет показана расширенная подсказка. Помимо этого у полей ввода можно задать подсказку ввода. Это свойство будет рассмотрено ниже, в разделе «Свойства поля» на стр. 307.

Свойства группы

Давайте познакомимся с некоторыми свойствами элемента формы Группа вида Обычная группа:

- Группировка – свойство может принимать значения: Горизонтальная, Горизонтальная если возможно и Вертикальная. Определяет, как будут группироваться подчиненные элементы. Стандартно новая группа создается с вариантом группировки Горизонтальная если возможно. При этом применяется горизонтальная группировка при наличии достаточного места по ширине формы. В противном случае элементы группируются вертикально.

- Отображение – свойство, оказывающее влияние на внешний вид группы. Свойство может принимать значения: Нет, Слабое выделение, Обычное выделение, Сильное выделение. В зависимости от варианта отображения группы различным образом выделяется заголовок группы, за показ которого отвечает свойство Отображать заголовок.

Например, в группу АдресаИКонтакты с горизонтальной группировкой, у которой отключен показ заголовка и свойство Отображение установлено в значение Нет, вложены две группы: Адрес и Контакты с вертикальной группировкой, у которых включен показ заголовка и свойство Отображение установлено в значение Слабое выделение (рис. 2.85).

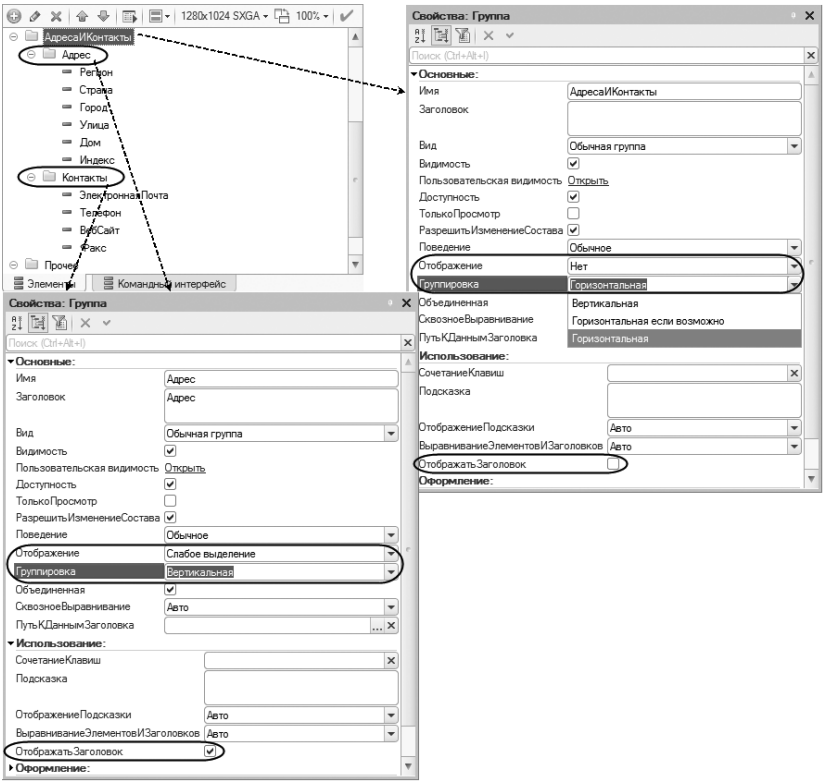


Рис. 2.85. Различные способы группировки и отображения групп

В результате группы в форме будут выглядеть следующим образом (рис. 2. 86).

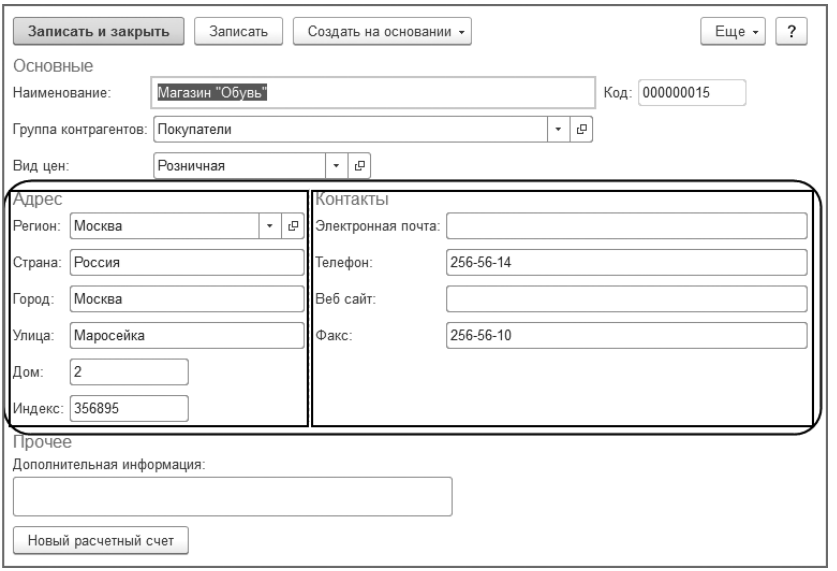


Рис. 2.86. Группировка и отображение элементов формы

Обычную группу можно настроить таким образом, что пользователь (в режиме 1С:Предприятие) сможет сворачивать или разворачивать такую группу. Например, можно разместить в сворачиваемой группе информацию, которая не важна при регулярной работе с формой, но иногда может потребоваться для просмотра и анализа.

Для этого нужно установить ряд свойств группы:

- Поведение – свойство, которое управляет поведением группы (Обычная или Свертываемая, Всплывающая).
- Свернута – состояние флажка определяет начальное состояние группы (свернута или развернута) при открытии формы.
- ОтображениеУправления – определяет элемент (Картинка или Гиперссылка заголовка), при нажатии на который пользователь может управлять состоянием группы (сворачивать/разворачивать). Стандартно свойство принимает значение Гиперссылка заголовка.

- ЗаголовокСвернутогоОтображения – заголовок, который будет отображаться в том случае, если группа находится в свернутом состоянии. Если свойство Свернутый заголовок не заполнено, то в свернутом состоянии будет отображаться обычный заголовок группы.

Например, в форме контрагента у группы Прочее с заголовком «Прочее» свойство Поведение установлено в значение Свертываемая, флажок Свернута установлен, свойство ОтображениеУправления установлено в значение Гиперссылка заголовка, а свойство ЗаголовокСвернутогоОтображения задано как «Прочее …» (рис. 2.87).

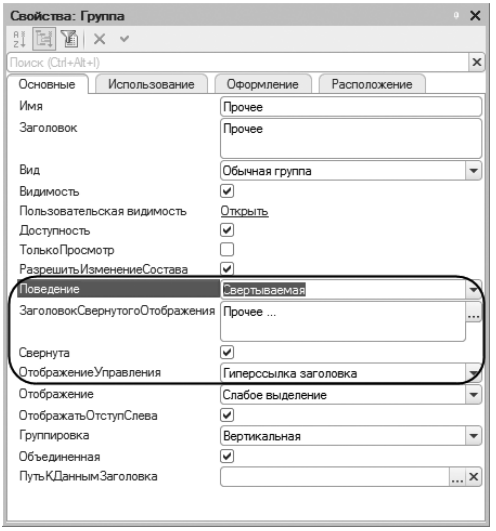


Рис. 2.87. Настройка свертываемой группы в конфигураторе

В результате при открытии формы контрагента в режиме 1С:Предприятие группа Прочее отображается в свернутом виде с заголовком (установленным в свойстве ЗаголовокСвернутогоОтображения) в виде гиперссылки. При нажатии на эту гиперссылку группа разворачивается и отображается содержимое группы под обычным заголовком, нажав на который, можно снова свернуть группу и спрятать дополнительную информацию (рис. 2.88).

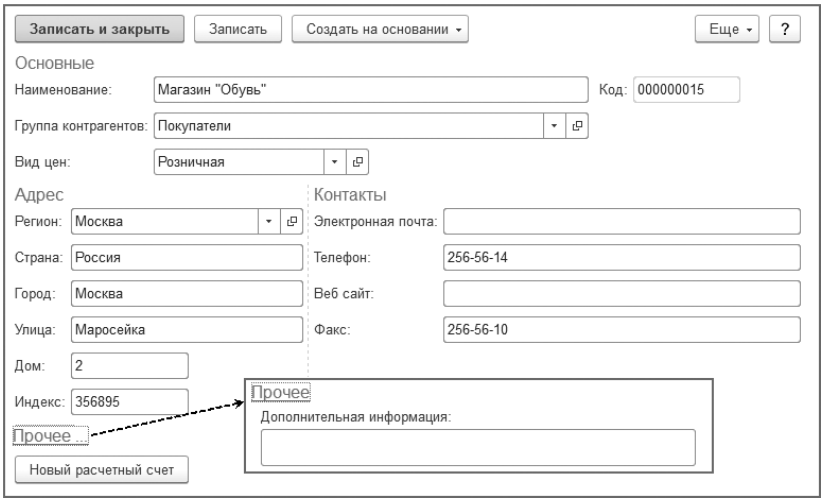


Рис. 2.88. Отображение свертываемой группы в режиме «1С:Предприятие»

Разновидностью свертываемой группы является всплывающая группа. В начальном состоянии всплывающая группа отображается в свернутом виде. При нажатии на заголовок такой группы она «всплывает» над родительской формой в специальном окне. В форме может существовать несколько всплывающих групп, но в один момент времени может быть открыта только одна из них.

Для такой группы свойство Поведение должно быть установлено в значение Всплывающая. Кроме того, у всплывающей группы должен быть задан заголовок. Заголовок выступает в роли текста гиперссылки, нажатие на которую приводит к отображению окна с содержимым группы (рис. 2.89).

Таким образом, всплывающие группы, с одной стороны, скрывая второстепенную информацию, делают форму более лаконичной. С другой стороны, в открытом состоянии они акцентируют внимание пользователя на своем содержимом.

Настройка всплывающих групп возможна начиная с версии платформы 8.3.12.

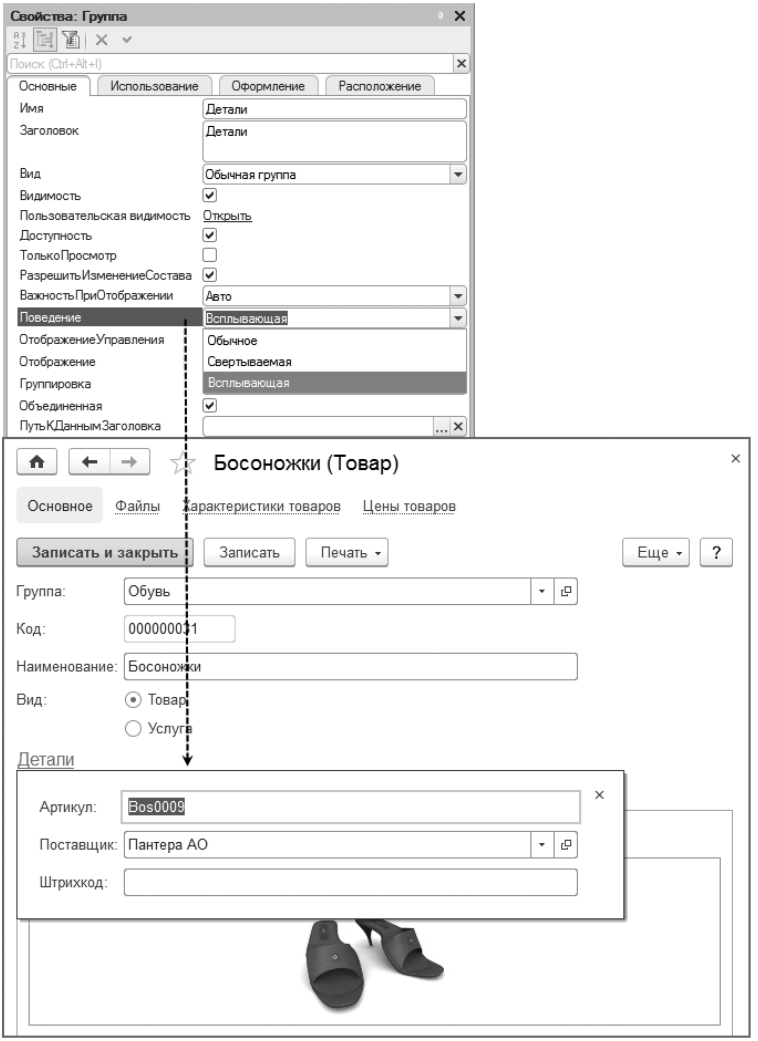


Рис. 2.89. Отображение всплывающей группы в режиме «1С:Предприятие»

Свойство ТолькоПросмотр влияет на все элементы, подчиненные группе.

Свойство группы РазрешитьИзменениеСостава управляет возможностью изменять состав группы в процессе настройки

формы пользователем. Если свойство выключено, то пользователь не может изменять состав и менять порядок элементов внутри группы. Однако свойство не влияет на возможность пользователя изменить видимость элементов, входящих в состав группы.

С помощью свойства ВыравниваниеЗаголовковИЭлементов задается тип выравнивания элементов и их заголовков внутри группы (а также страницы или самой формы). Свойство может принимать значения: Авто (значение по умолчанию), Нет, Элементы лево, заголовки лево, Элементы право, заголовки лево, Элементы право, заголовки право, Элементы лево, заголовки право, Элементы авто, заголовки лево.

Для того чтобы выровнять элементы из разных групп вдоль воображаемой опорной линии формы, используется свойство группы СквозноеВыравнивание. Это свойство может принимать значения: Авто, Использовать, Не использовать. Стандартно данное свойство устанавливается в значение Авто. Это значит, что сквозное выравнивание определяется платформой самостоятельно.

Явное указание для данного свойства значения Не использовать приведет к тому, что в таком группирующем элементе не будет выполняться выравнивание с учетом элементов, не входящих в текущую группу. Сквозное выравнивание элементов используется только для групп с обычным поведением.

Все новые группы стандартно создаются с установленным свойством Объединенная. С помощью этого свойства можно создавать группы, состоящие из нескольких колонок и строк одновременно, с поддержкой выравнивания элементов во всех колонках и строках:

- свойство установлено – группа считается единым элементом;
- свойство сброшено – каждый элемент, входящий в состав группы со сброшенным свойством, считается отдельным элементом.

Для элемента формы Группа вида Страницы интерес представляет свойство ОтображениеСтраниц (рис. 2.90). Данное свойство управляет положением закладок страницы. Если используется вариант свойства Нет, то при наличии нескольких

Пример использования свойства Объединенная рассмотрен в разделе «Выравнивание между группами» стр. 415.

страниц их переключение можно осуществлять только средствами встроенного языка.

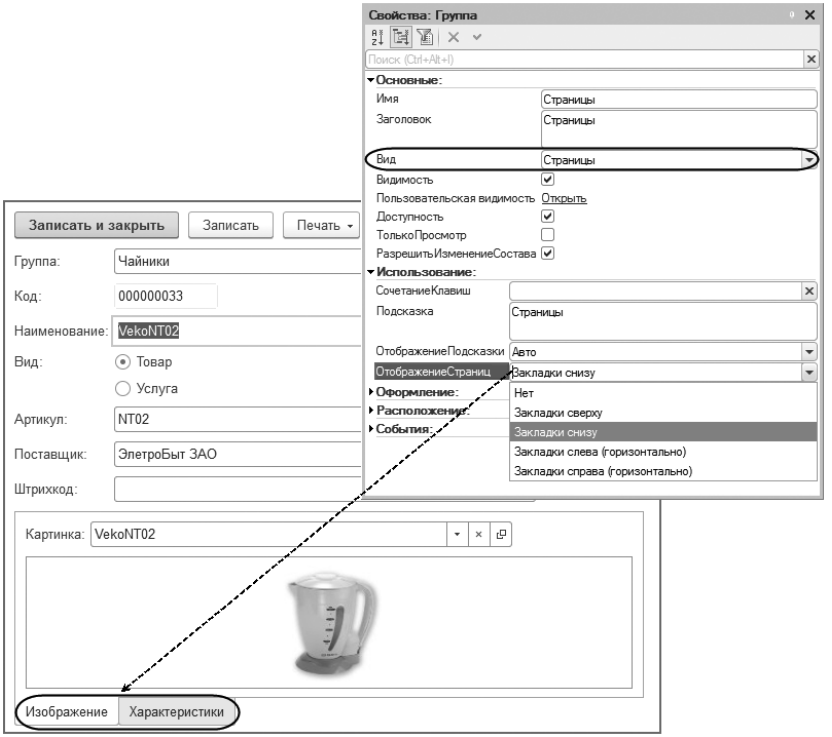


Рис. 2.90. Свойство «Отображение страниц»

Если у группы вида Страница или Обычная группа указано свойство ПутьКДаннымЗаголовка, то эти данные будут автоматически отображаться в заголовке группы после самого заголовка, если он задан (рис. 2.91).

У элемента формы Группа вида Страница или Обычная группа, а также у самой формы есть свойства ГоризонтальныйИнтервал и ВертикальныйИнтервал. Изменяя эти свойства, можно добиться необходимой плотности элементов формы (а также отдельной группы или страницы) как по вертикали, так и по горизонтали.

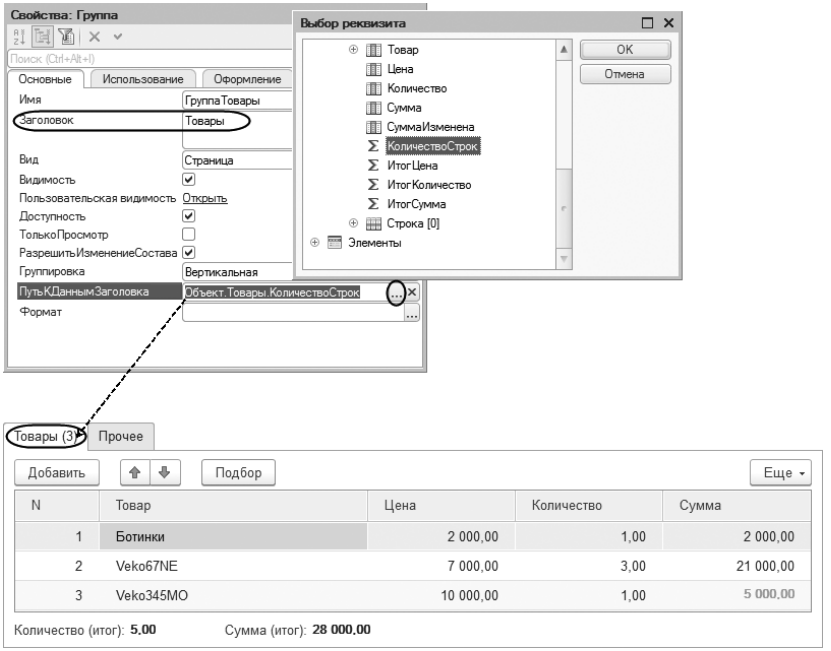


Рис. 2.91. Свойство «Путь к данным заголовка»

Свойства принимают значения: Нет, Авто, Половинный, Одинарный, Полуторный, Двойной. Значение Авто, установленное у любого из этих свойств, означает, что интервал будет определяться по родительскому группирующему элементу. Для формы значение интервала Авто определяет платформа.

С помощью свойств ГоризонтальноеПоложениеПодчиненных, ВертикальноеПоложениеПодчиненных (Авто, Лево, Центр, Право) можно управлять относительным расположением элементов обычной группы (а также страницы или самой формы).

У элемента формы Группа вида Командная панель существует свойство Источник команд, с помощью которого можно указать элемент формы, который будет предоставлять «свои» команды для отображения в командной панели.

Источником команд может быть как сама форма, так и элементы формы типа Таблица, и поля вида Поле табличного документа, Поле форматированного документа, Поле графической схемы, Поле планировщика. Состав команд для отображения в командной панели регулируется свойством Состав команд элемента формы, являющегося источником команд (рис. 2.92).

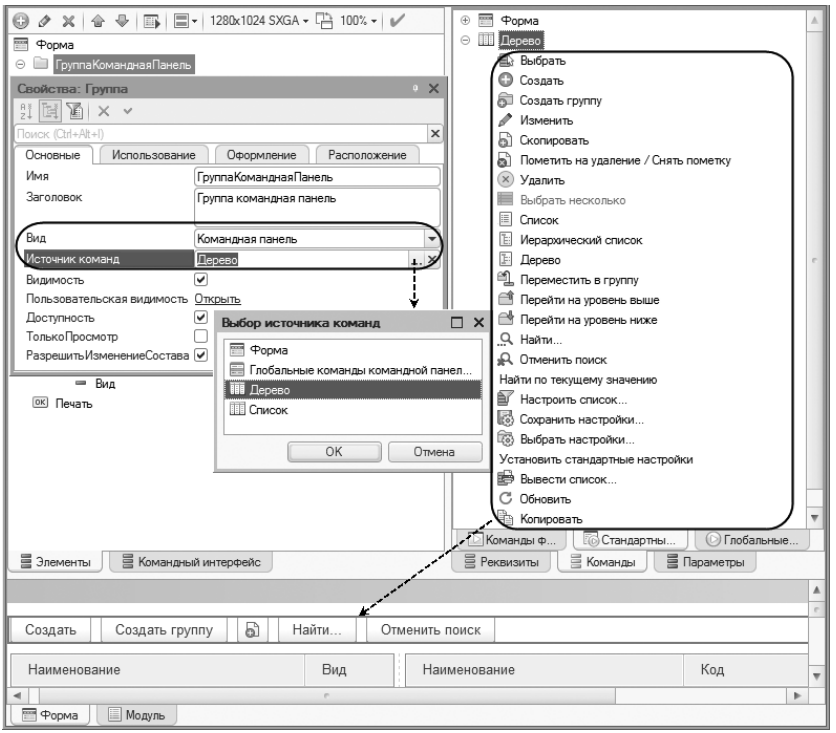


Рис. 2.92. Свойство «Источник команд» командной панели

Свойство ГоризонтальноеПоложение позволяет управлять выравниванием кнопок в командной панели. Возможны значения: Лево, Право, Центр.

Если в командную панель входит подчиненная группа вида Группа кнопок, то с помощью свойства Отображение можно задать Обычное или Компактное отображение группы кнопок в командной панели.

При компактном отображении кнопки в группе кнопок располагаются вплотную друг к другу (рис. 2.93).

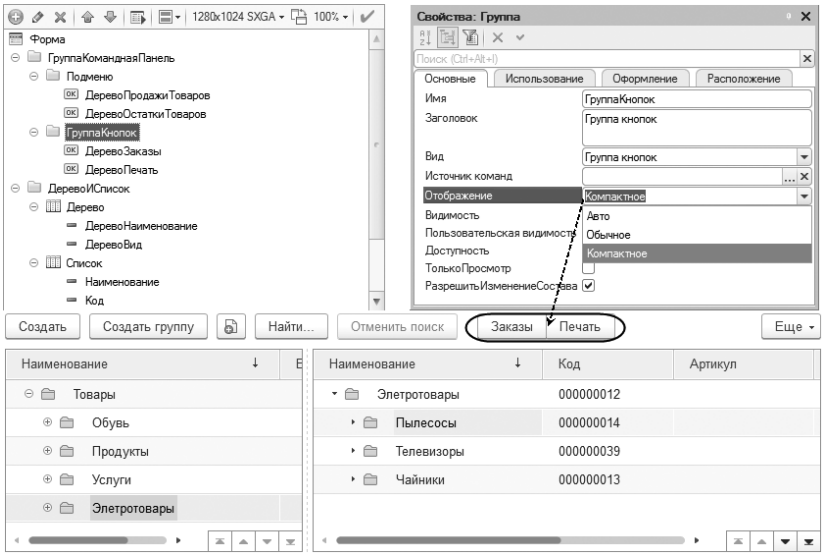


Рис. 2.93. Свойство «Отображение» группы кнопок

Свойства поля

Наибольший интерес (в плане разнообразия свойств и частоты использования) представляет элемент формы Поле вида Поле ввода.

Свойство ВыбиратьТип влияет на поведение системы при выборе типа данных реквизита, который отображает поле ввода. Проявляется это в случае использования реквизита составного типа данных (рис. 2.94).

Свойство Маска позволяет задать маску для ввода символов. Например, можно задать маску для ввода телефона (рис. 2.95).

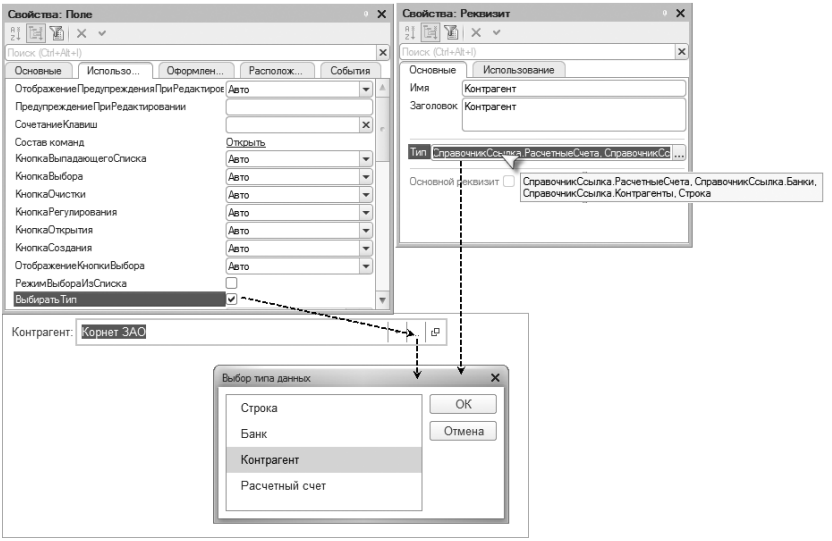


Рис. 2.94. Свойство «Выбирать тип»

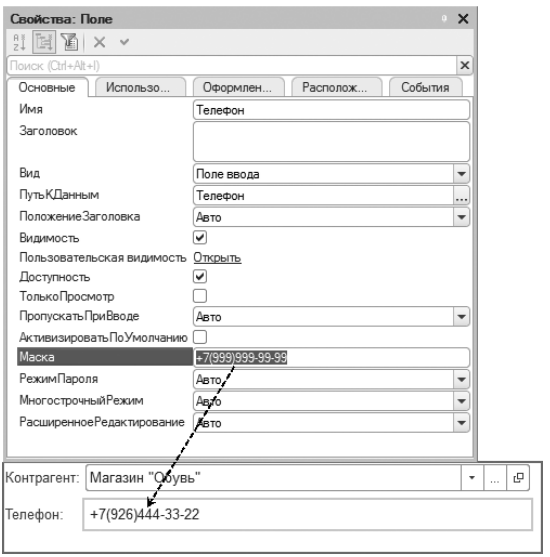


Рис. 2.95. Свойство «Маска»

Свойство РежимПароля при установке значения Да будет заменять вводимые символы специализированными символами (рис. 2.96).

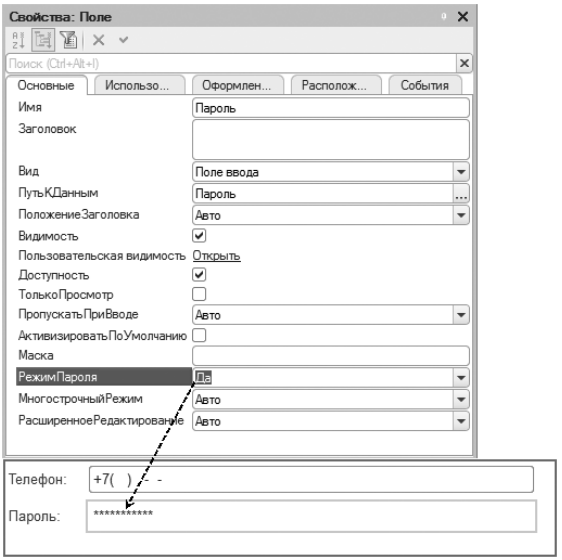


Рис. 2.96. Свойство «Режим пароля»

Свойство СписокВыбора позволяет задать список, из которого можно выбирать значения в поле ввода. Для того чтобы поле ввода работало как поле выбора, нужно установить свойство РежимВыбораИзСписка. В этом случае кнопка выпадающего списка появится у поля автоматически, хотя стандартно в поле ввода она не показывается (рис. 2.97).

Если же флажок у свойства РежимВыбораИзСписка снят, то можно явно задать наличие кнопки выпадающего списка у поля ввода, установив свойство КнопкаВыпадающегоСписка в значение Да. Хотя сам список выбора все равно будет открываться при переходе к полю и без кнопки выпадающего списка, но многим пользователям с ней привычнее и сразу понятно, что это поле для выбора из списка значений.

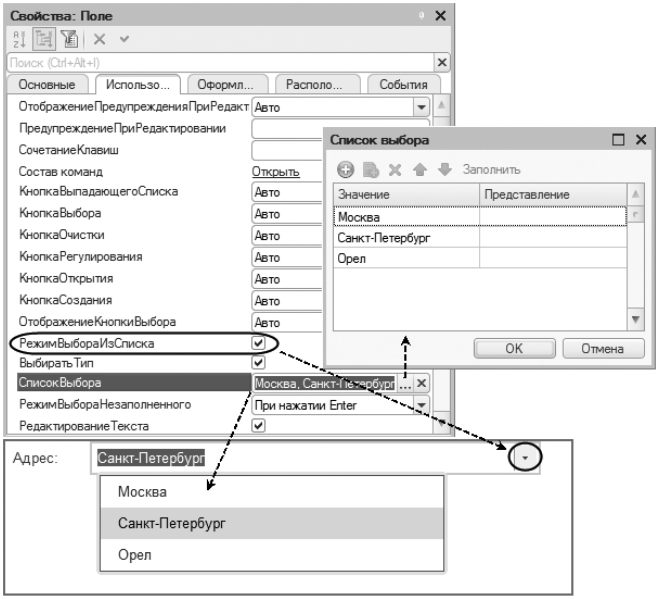


Рис. 2.97. Свойство «Список выбора»

Для элементов формы, отображающих примитивные данные типа Число, Дата, Булево, свойство ФорматРедактирования позволяет задать формат вводимых данных с помощью «Конструктора форматной строки» (рис. 2.98).

Свойства Минимальное значение и Максимальное значение позволяют ограничить ввод числовых значений. При этом системой будет контролироваться соответствие вводимого числа заданным критериям.

Свойство Выбор групп и элементов позволяет ограничить выборку данных, например, только элементами иерархического справочника.

Свойство ПредупреждениеПриРедактировании содержит строку, выдаваемую в качестве предупреждения при начале редактирования поля формы в том случае, если ОтображениеПредупрежденияПриРедактировании установлено в значение Отображать.

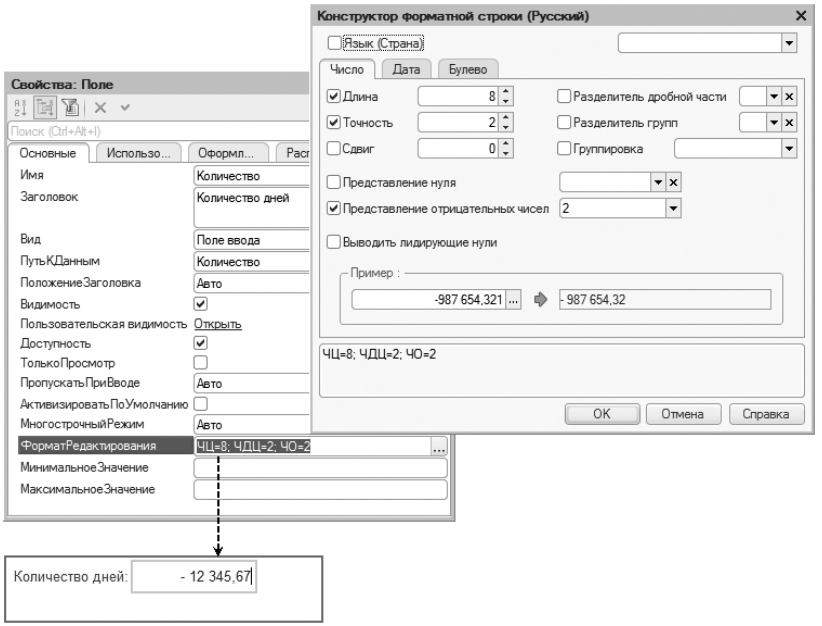


Рис. 2.98. Свойство «Формат редактирования»

Если значение свойства ОтображениеПредупрежденияПриРедактировании равно Отображать, а свойство ПредупреждениеПриРедактировании не задано, то строка предупреждения формируется системой автоматически.

Если значение свойства ОтображениеПредупрежденияПриРедактировании равно Авто, то для стандартных реквизитов Код и Номер будет использоваться значение Отображать, а для остальных полей – Не отображать.

Свойство БыстрыйВыбор позволяет выбирать значения ссылочных типов из выпадающего списка, а не из формы выбора. Например, это могут быть элементы неиерархического справочника, заведомо имеющего небольшой список. Обычно это свойство устанавливается на уровне прикладного объекта конфигурации или реквизита, отображаемого полем, но если у «вышестоящих» объектов это свойство не установлено или его действие необходимо переопределить,

то можно воспользоваться свойством БыстрыйВыбор элемента формы.

Подробнее о свойстве БыстрыйВыбор реквизитов и прикладных объектов конфигурации рассказывается в разделе «Быстрый выбор» на стр. 336.

Свойство ИсторияВыбораПриВводе позволяет сохранять и отображать историю выбора при вводе ссылочных значений в поле формы. История выбора состоит из наиболее часто используемых и последних выбранных значений данного ссылочного типа. Обычно это свойство устанавливается на уровне прикладного объекта конфигурации или реквизита, отображаемого элементом. Но если действие свойства необходимо переопределить в конкретной форме, то можно воспользоваться свойством ИсторияВыбораПриВводе элемента формы.

ПОДРОБНЕЕ

Подробнее о свойстве ИсторияВыбораПриВводе реквизитов и прикладных объектов конфигурации рассказывается в разделе «История выбора при вводе» на стр. 338.

Свойство СвязиПараметровВыбора позволяет задать значение отбора для возможных вариантов выбора значений ссылочного поля. При установке связи параметров выбора значение, выбранное в одном поле формы, накладывает ограничение на выбор значений в другом поле. Этот отбор будет применяться при открытии формы выбора, при отображении списка быстрого выбора и при выполнении ввода по строке.

ПОДРОБНЕЕ

Подробнее о свойстве реквизитов СвязиПараметровВыбора рассказывается в разделе «Связи параметров выбора» на стр. 340.

Свойство ПараметрыВыбора позволяет указать значения параметров, которые будут применяться при выборе значения поля. Но этот выбор будет выполняться не на основании значений других реквизитов формы, а на основании условий, наложенных на свойства

самого выбираемого объекта. Отбор можно наложить на все реквизиты объекта конфигурации, на который ссылается данное поле. Этот отбор будет применяться при открытии формы выбора, при отображении списка быстрого выбора и при выполнении ввода по строке.

Подробнее о свойстве реквизитов ПараметрыВыбора рассказывается в разделе «Параметры выбора» на стр. 342.

Как правило, такие ограничения бизнес-логики, как ограничения выбора, должны быть одинаковыми для всех форм, в которых редактируется тот или иной объект. Поэтому задавать параметры выбора и связи параметров выбора рекомендуется в свойствах объектов конфигурации – в реквизитах справочников, документов и т. п. Однако могут встречаться случаи, когда ограничения выбора могут зависеть от конкретного сценария работы. В таких случаях параметры выбора могут быть уточнены по месту, в конкретной форме.

Свойство КнопкаСоздания определяет наличие кнопки для создания нового элемента в выпадающем списке, открывающемся под ссылочным полем. Обычно возможность создания при вводе устанавливается на уровне прикладного объекта конфигурации или реквизита, отображаемого полем, – у них это свойство называется СозданиеПриВводе. Но если у «вышестоящих» объектов это свойство не установлено или его действие необходимо переопределить, то можно воспользоваться свойством КнопкаСоздания элемента формы.

ПОДРОБНЕЕ

Подробнее о свойстве СозданиеПриВводе реквизитов и прикладных объектов конфигурации рассказывается в разделе «Создание при вводе» на стр. 344.

Свойство СвязьПоТипу позволяет указать элемент формы, которым будет определяться тип значения выбираемых в поле данных, ограничивая тип вводимых значений для поля ввода. Настройка связи по типу имеет смысл для реквизитов с составным типом данных, логически связанных с другим реквизитом. Например, когда требуется задать

связь типа значения характеристики объекта с видом этой характеристики.

Подробнее о свойстве реквизитов СвязьПоТипу рассказывается в разделе «Связь по типу» на стр. 346.

Свойство ФормаВыбора устанавливает форму, которая будет применяться для выбора значения реквизита, отображаемого в поле. Используется для реквизитов, тип которых образован объектом конфигурации, имеющим подчиненные формы. Стандартно для выбора ссылочного реквизита используется форма, назначенная в качестве основной формы выбора для объекта конфигурации, на который он ссылается. Но с помощью свойства ФормаВыбора можно установить для реквизита другую форму, которая будет вызываться при выборе значений в связанном с ним поле.

Собственную форму выбора, как и связь по типу (описанную выше), также рекомендуется задавать в свойствах самого реквизита, но, если требуется, эти свойства можно переопределить в конкретной форме, в свойствах поля ввода, связанного с реквизитом.

ПРИМЕЧАНИЕ

Про свойства реквизитов объектов конфигурации, которые влияют на аналогичные свойства связанных с ними полей ввода, рассказывается в разделе «Интерфейсные свойства реквизитов объектов конфигурации» на стр. 334.

Свойство поля ввода ПодсказкаВвода позволяет указывать подсказку, которая будет формироваться непосредственно «внутри» поля ввода, если тип связанного с полем реквизита не является типом Число или Дата и значение связанного реквизита не содержит значение по умолчанию для данного типа (рис. 2.99).

Для полей вида Поле флажка и Поле переключателя существует возможность указывать вид отображения. Это делается с помощью свойства ВидФлажка/ВидПереключателя у соответствующего элемента формы. Можно выбрать представление в виде флажка/переключателя или тумблера (рис. 2.100).

Для поля вида Поле картинки интерес представляет свойство Масштабировать. Если флажок свойства установлен, то для картинки,

не помещающейся в области размещения, появляются линейки прокрутки. Изменить размер картинки можно с помощью команд контекстного меню.

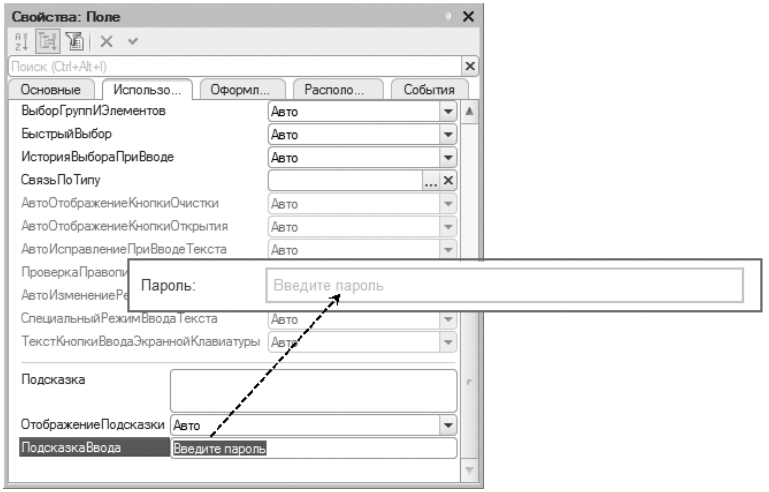


Рис. 2.99. Свойство «Подсказка ввода»

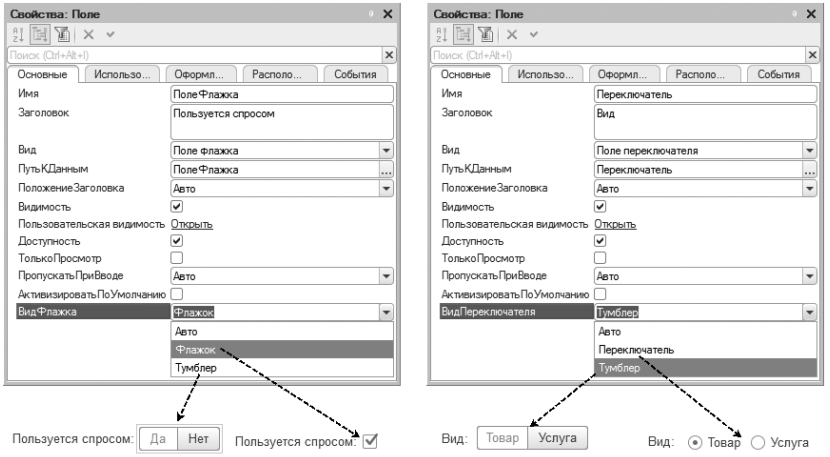


Рис. 2.100. Свойства «Вид флажка» и «Вид переключателя»

С помощью свойства ТекстНевыбраннойКартинки можно задать текст, отображаемый, если картинка не выбрана.

Для поля вида Поле индикатора с помощью свойства Отображение можно управлять вариантом заполнения индикатора. Возможны стили заполнения индикатора Плавное, Прерывистое и Прерывистое наклонное.

С помощью свойства ОтображатьПроценты можно показывать процент заполнения индикатора (рис. 2.101).

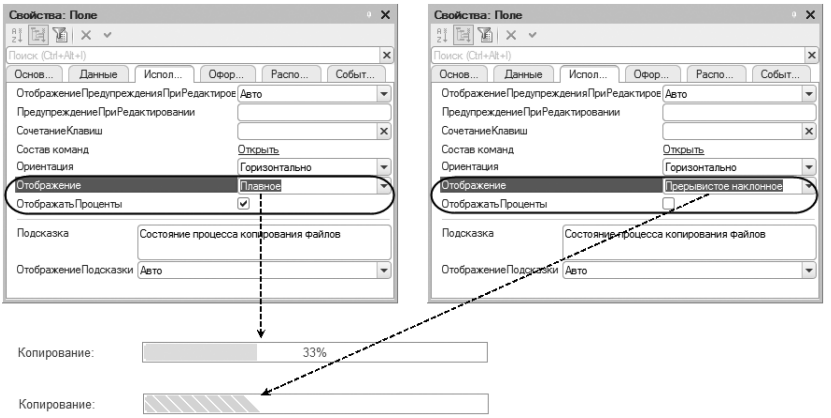


Рис. 2.101. Свойства поля индикатора

Для поля вида Поле полосы регулирования с помощью свойства ОтображениеРазметки можно управлять вариантом расположения разметки. Возможны значения: Не отображать, Сверху (слева), Снизу (справа), С обеих сторон.

С помощью свойства ШагРазметки можно задать значение шага, с которым будет отображаться разметка полосы регулирования.

Для поля вида Поле табличного документа можно управлять наличием полос прокрутки у табличного документа с помощью свойств ВертикальнаяПолосаПрокрутки/ГоризонтальнаяПолосаПрокрутки. Возможны значения:

- Не использовать – полосы прокрутки не используются;

- Использовать всегда – полосы прокрутки всегда присутствуют в табличном документе;
- Использовать автоматически – полосы прокрутки используются в табличном документе только тогда, когда содержимое табличного документа по соответствующему измерению не помещается в видимой области. Это значение устанавливается по умолчанию.

С помощью свойства Редактирование можно задать возможность изменения табличного документа. С помощью свойства ОтображатьСетку можно управлять отображением сетки в поле табличного документа (рис. 2.102).

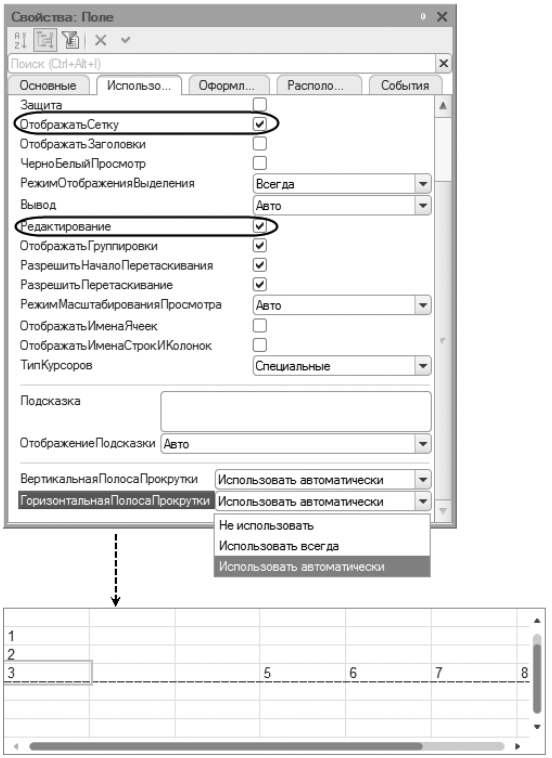


Рис. 2.102. Свойства поля табличного документа

Для поля вида Поле календаря с помощью свойства РежимВыделения можно управлять режимом выделения дат. Возможны значения: Одиночный, Множественный, Интервал (выделение непрерывного интервала дат).

С помощью свойства ОтображатьПанельМесяцев можно показать в поле календаря панель месяцев, которая повышает быстроту и удобство выбора даты из календаря.

Поля формы вида Поле табличного документа, Поле форматированного документа, Поле графической схемы, Поле планировщика являются источником стандартных команд формы. Поэтому рядом с ними можно добавить группу вида Командная панель и выбрать соответствующие поля в качестве источника команд этой панели. Состав команд этих полей регулируется с помощью свойства Состав команд (рис. 2.103, 2.104).

Команды, отключенные при редактировании данного свойства, становятся недоступными.

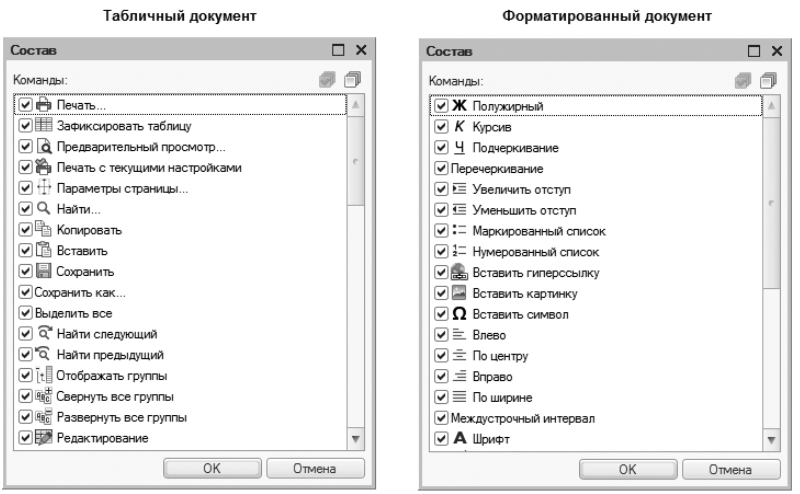


Рис. 2.103. Состав стандартных команд полей табличного и форматированного документа

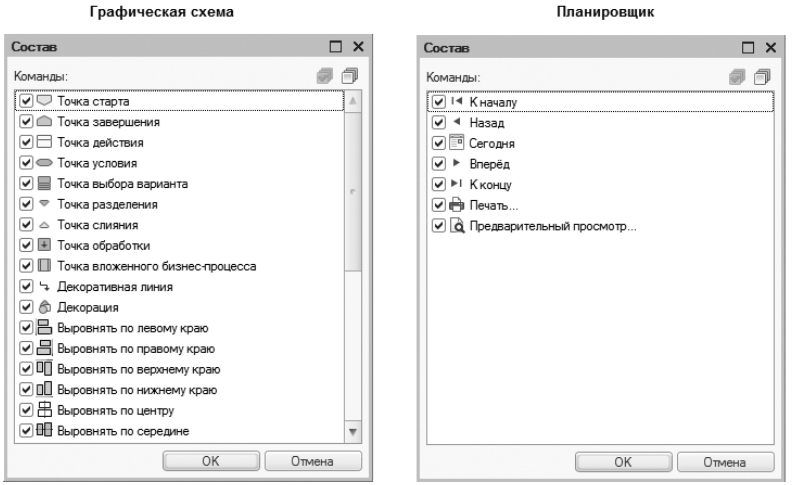


Рис. 2.104. Состав стандартных команд полей графической схемы и планировщика

Свойства таблицы

Таблица формы является источником стандартных команд, поэтому у таблицы существует собственная командная панель, в которой располагаются эти команды. Состав команд таблицы регулируется с помощью свойства Состав команд (рис. 2.105).

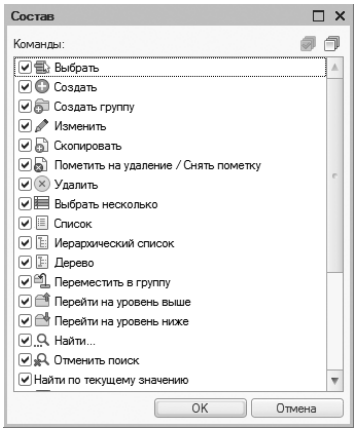


Рис. 2.105. Состав стандартных команд таблицы

Команды, отключенные при редактировании данного свойства, становятся недоступными.

Наличие и расположение командной панели таблицы определяется свойством ПоложениеКоманднойПанели. Возможны значения: Нет, Авто, Верх, Низ. В форме списка, в форме выбора командная панель таблицы обычно не отображается, поскольку все команды собираются в командную панель формы.

Свойство Отображение позволяет установить вариант отображения данных. Возможны значения:

- Список. Данные отображаются в виде обычного (плоского) списка. Если справочник иерархический, то группы расположены вперемешку с элементами (рис. 2.106).

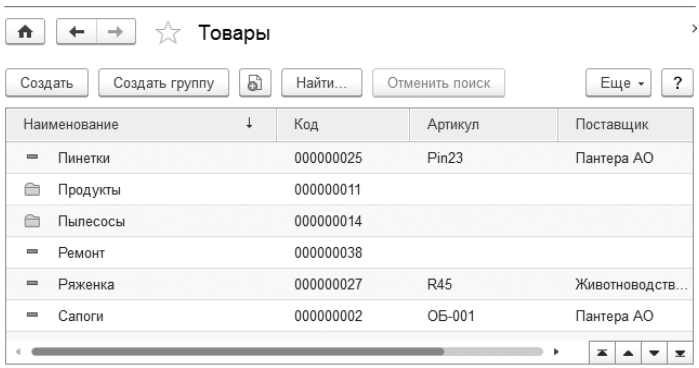


Рис. 2.106. Отображение таблицы в виде списка

- Иерархический список. Данные отображаются в виде списка с учетом иерархии групп и элементов (рис. 2.107).
- Дерево. Данные отображаются в виде дерева (рис. 2.108). Если таблица связана с реквизитом типа ДинамическийСписок, у которого в настройках списка (свойство реквизита формы Настройка списка) задана группировка, то список всегда будет отображаться в режиме Дерево.

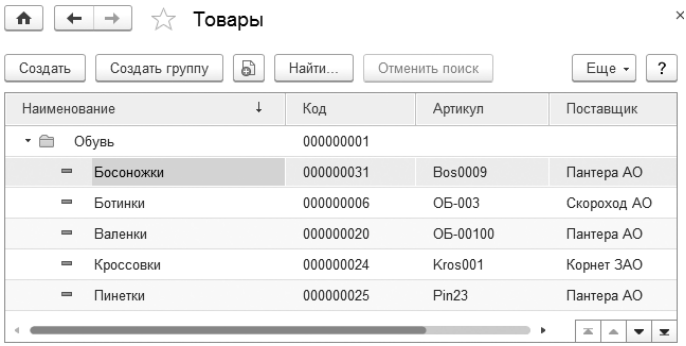


Рис. 2.107. Отображение таблицы в виде иерархического списка

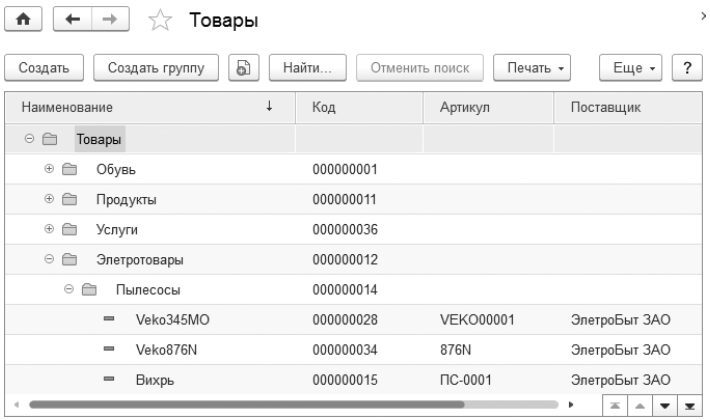


Рис. 2.108. Отображение таблицы в виде дерева

Для иерархических списков не рекомендуется устанавливать свойство НачальноеОтображениеДерева в значение Раскрывать все уровни, так как это приведет к существенному снижению скорости открытия больших списков. Следует использовать значения Не раскрывать или Раскрывать верхний уровень.

Свойство Вывод управляет возможностями сохранения, печати и копирования данных, отображаемых элементом формы Таблица.

Если таблица отображает данные динамического списка, то с помощью свойства ОбновлениеПриИзмененииДанных можно

управлять обновлением списка при интерактивном добавлении или редактировании данных. Свойство Автообновление содержит признак необходимости автообновления данных в таблице через интервал, заданный в свойстве ПериодАвтообновления.

С помощью свойства ФиксацияВТаблице (Лево, Право, Нет) можно зафиксировать некоторые столбцы в таблице (отменить их прокрутку). Например, в табличной части документа можно зафиксировать слева колонку Номер и Товар, а колонку Сумма прижать к правому краю таблицы (рис. 2.109).

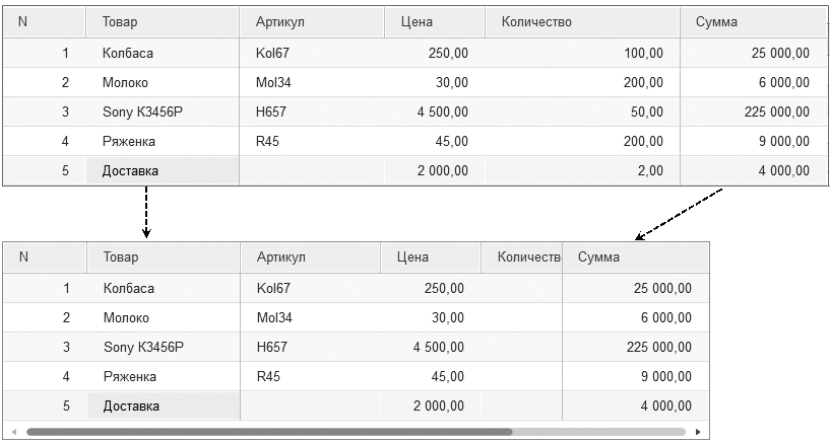


Рис. 2.109. Фиксация столбцов в таблице

В том случае, если область фиксации таблицы формы больше видимой области таблицы, выполняется автоматическое отключение фиксации колонок. Если признак фиксации указан у группы колонок, то прижата будет вся группа (вместе со всеми подчиненными элементами формы), при этом значение свойства ФиксацияВТаблице для подчиненных элементов игнорируется.

Управлять наличием полос прокрутки у таблицы можно с помощью свойств ВертикальнаяПолосаПрокрутки/ГоризонтальнаяПолосаПрокрутки. Возможны значения:

- Не использовать – полосы прокрутки не используются;

- Использовать всегда – полосы прокрутки всегда присутствуют в таблице формы;
- Использовать автоматически – полосы прокрутки используются в таблице только тогда, когда содержимое таблицы по соответствующему измерению не помещается в видимой области. Это значение устанавливается по умолчанию.

С помощью свойства ВариантУправленияВысотойТаблицы можно управлять высотой таблицы формы. Возможны следующие значения:

- В строках формы – в этом случае высота таблицы задается с помощью свойства Высота таблицы формы. Для ограничения максимальной высоты таблицы используются свойства АвтоМаксимальнаяВысота и МаксимальнаяВысота.
- В строках таблицы – в этом случае высота таблицы задается с помощью свойства ВысотаВСтрокахТаблицы таблицы формы. Для ограничения максимальной высоты таблицы используются свойства АвтоМаксимальнаяВысотаВСтрокахТаблицы и МаксимальнаяВысотаВСтрокахТаблицы.
- По содержимому – если таблица связана с динамическим списком, то высота таблицы задается значением свойства ВысотаВСтрокахТаблицы. Если значение этого свойства равно 0, то высота таблицы определяется значением свойства Высота.

Если таблица не отображает данные динамического списка, то, установив свойство ВариантУправленияВысотойТаблицы в значение По содержимому, можно разработать форму, в которой таблица будет отображать то количество строк, которое в ней находится, тем самым уменьшив количество полос прокрутки в форме и сделав саму форму более компактной.

Свойства кнопки

Свойство ТолькоВоВсехДействиях (Да, Нет, Авто) определяет, будет ли отображаться кнопка еще где-нибудь (в местах, определенных разработчиком формы) кроме подменю Еще.

Если у кнопки установлено свойство Пометка, она отображается как нажатая.

Свойство Отображение определяет вариант отображения кнопки на экране (Авто, Текст, Картинка, Картинка и текст).

Если выбрано отображение Картинка и текст, то картинка, расположенная на кнопке, может располагаться слева или справа относительно текста кнопки. Для управления этим расположением предназначено свойство кнопки ПоложениеКартинки (Право, Лево, Авто). Если данное свойство установлено в значение Авто, то картинка будет располагаться с левой стороны текста (см. рис. 2.110).

С помощью свойства Фигура (Авто, Обычная, Овальная) можно задать обычную или овальную фигуру для отображения кнопки (см. рис. 2.110).

С помощью свойства ОтображениеФигуры можно указать несколько различных способов отображения фигуры кнопки (рис. 2.110):

- Авто – в этом случае для стандартных кнопок используется вариант отображения, заданный для команды. В остальных случаях трактуется как Всегда;
- Всегда – в этом случае фигура кнопки отображается;
- При активности – в этом случае в обычном состоянии кнопка отображается без фона и рамки. В том случае, если курсор мыши указывает на кнопку, на ней установлен фокус ввода или у кнопки установлено свойство Пометка – отображается стандартная кнопка с заданными цветами рамки и фона;
- Нет – в этом случае в обычном состоянии кнопка отображается без фона и рамки. При наведении курсора у заголовка кнопки появляется подчеркивание и изменяется курсор мыши.

Свойство ОтображениеФигуры не действует на команды, расположенные в подменю.

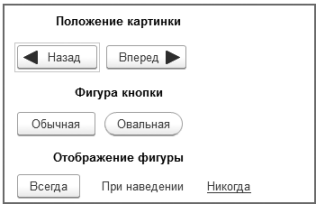


Рис. 2.110. Свойства кнопки

Кроме того, для кнопки так же, как и для других элементов формы, можно задать свойства: ЦветФона, ЦветТекста, ЦветРамки и Шрифт.

Как добавить новые элементы формы

Для того чтобы добавить новый элемент формы, необходимо воспользоваться одним из способов, предоставляемых редактором формы.

- Воспользоваться конструктором форм. Работа конструктора рассматривается в главе 2.2 на стр. 242. При таком способе будет создана новая форма, однако если объекту конфигурации в процессе разработки после создания формы был добавлен новый реквизит или табличная часть, то эти изменения можно будет учесть.
- Воспользоваться кнопкой командной панели области Элементы редактора формы (рис. 2.111).

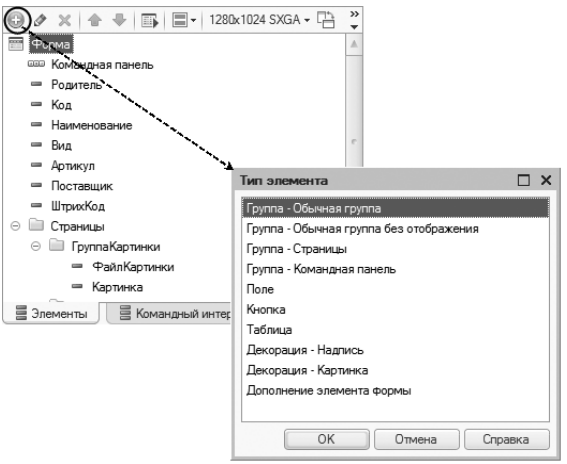


Рис. 2.111. Кнопка командной панели для добавления элементов формы

- Воспользоваться контекстным меню области Элементы редактора формы (рис. 2.112).
- Использовать возможности перетаскивания из других окон с помощью мыши.

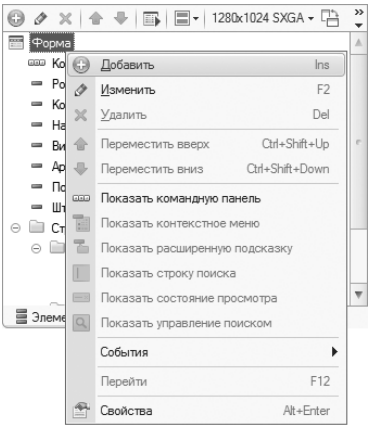


Рис. 2.112. Кнопка контекстного меню для добавления элементов формы

Непосредственно перед добавлением нового элемента формы можно спозиционироваться на элементе верхнего уровня, в подчинение которому планируется поместить новый элемент. Перемещать элементы формы по дереву иерархии можно с помощью мыши (перетаскивание) или кнопок Переместить вверх, Переместить вниз.

Таким образом, после того как форма создана конструктором, проще всего добавлять новые элементы формы путем перетаскивания мышью нужных реквизитов и команд в дерево элементов формы. При этом создается элемент формы «подходящего» вида и в нем автоматически устанавливается связь с отображаемыми данными.

Кроме того, новые элементы формы можно добавить кнопкой командной панели или кнопкой контекстного меню редактора, но в этом случае нужно не забыть вручную связать их с одним из реквизитов формы.

Глава 2.4. Влияние объектов конфигурации на форму

Объекты конфигурации определяют не только функциональность будущей формы через ее основной реквизит, но и оказывают влияние на внешний вид формы, на поведение ее элементов. На форму влияют как свойства самого объекта конфигурации, так и свойства его реквизитов.

Заголовок формы

Прежде всего, с помощью свойств объектов конфигурации формируется заголовок формы. Однако такое влияние на заголовок формы оказывается не всегда. У разработчика приложения остается возможность ручного формирования заголовка формы.

Рассмотрим небольшой пример, в котором будут задействованы свойства объекта конфигурации, оказывающие влияние на формирование заголовка формы объекта (формы элемента справочника, формы документа, формы задачи).

По умолчанию для вновь созданной формы ее заголовок формируется полностью в автоматическом режиме. Это означает, что свойство формы Заголовок не заполнено, а флажок АвтоЗаголовок установлен. При такой установке свойств на автоматически формируемый заголовок формы оказывает влияние свойство объекта конфигурации Синоним (рис. 2.113).

Так происходит потому, что после добавления объекта конфигурации платформа автоматически сразу же формирует синоним объекта на основе его имени. Как правило, разработчик редактирует этот синоним, чтобы он содержал четкое и краткое описание объекта.

При создании нового объекта его синоним показывается в заголовке формы – при условии, что представления объекта не заданы. Если задано расширенное представление объекта, то в заголовке формы будет показано именно оно, а не синоним. Речь о представлениях объекта и списка пойдет ниже.

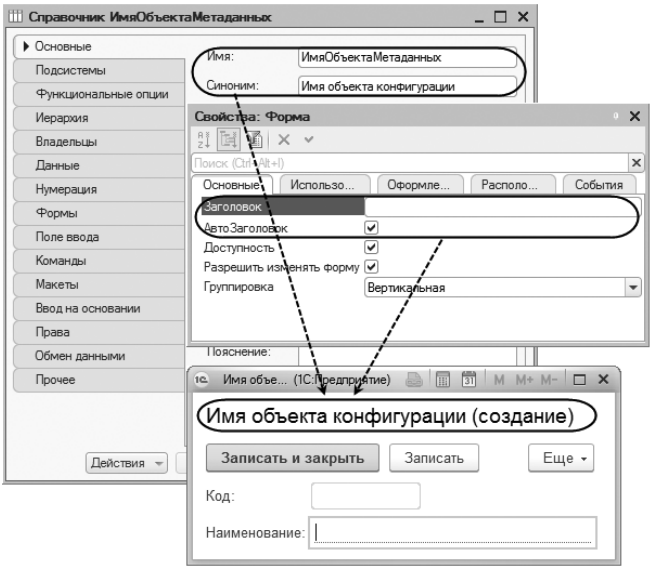


Рис. 2.113. Формирование заголовка формы по свойству «Синоним»

После записи такого элемента в информационную базу в формировании заголовка формы участвует еще и свойство объекта Основное представление. Например, для элемента справочника, если свойство имеет значение В виде наименования, то в заголовке формы будет показано его наименование. А если В виде кода, то в заголовке формы будет показан код элемента (рис. 2.114).

У объектов конфигурации существует ряд интерфейсных свойств, влияющих на заголовок формы в пользовательском режиме работы. Нами будут рассмотрены свойства Представление объекта, Расширенное представление объекта, Представление списка и Расширенное представление списка.

При формировании заголовка формы объекта, если свойство Представление объекта заполнено, приоритет отдается именно ему (рис. 2.115). Свойство задается в единственном числе и пока-

Согласно методикам разработки фирмы «1С» свойство Представление объекта заполняется в том случае, если Синоним объекта не представляет собой краткое и четкое описание объекта в единственном числе.

зывается как название одного объекта, например: Контрагент для справочника Контрагенты, Организация для справочника Организации, Договор для справочника Договоры. Свойство участвует в формировании названия команды в интерфейсе приложения.

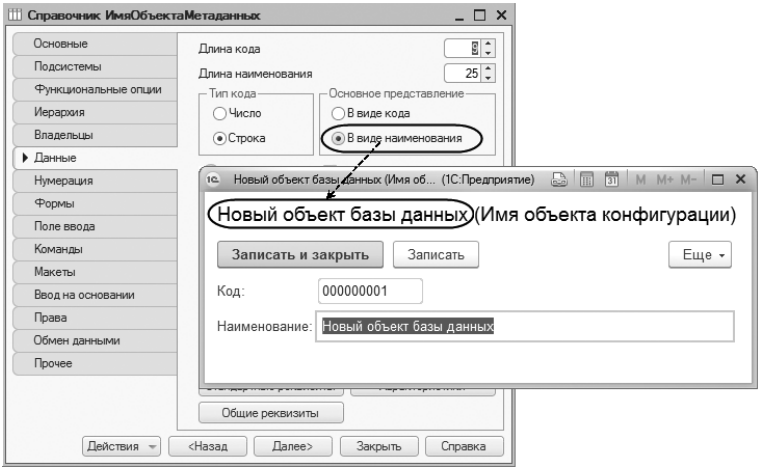


Рис. 2.114. Заголовок формы после записи элемента

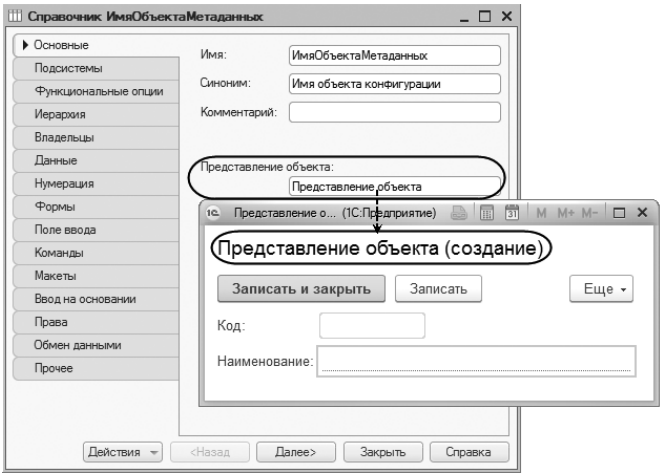


Рис. 2.115. Формирование заголовка формы по свойству «Представление объекта»

Согласно методикам разработки фирмы «1С» свойство Расширенное представление объекта используется в том случае, если Представление объекта или его Синоним не дают полного представления о наименовании объекта. Свойство заполняется в единственном числе.

В случае использования свойства Расширенное представление объекта в заголовке формы отображается именно оно (рис. 2.116). Остальные рассмотренные свойства (ПредставлениеОбъекта, Синоним) могут использоваться платформой только в том случае, если расширенное представление объекта не задано.

Таким образом, можно сказать, что свойство Представление объекта отвечает за формирование команд, а Расширенное представление объекта – за отображение в заголовке формы. Например: «Товар» и «Карточка товара». Но если эти представления совпадают, тогда можно задать только представление объекта, и оно же автоматически будет использоваться в заголовке формы. С представлениями списка используется тот же принцип (рис. 2.117).

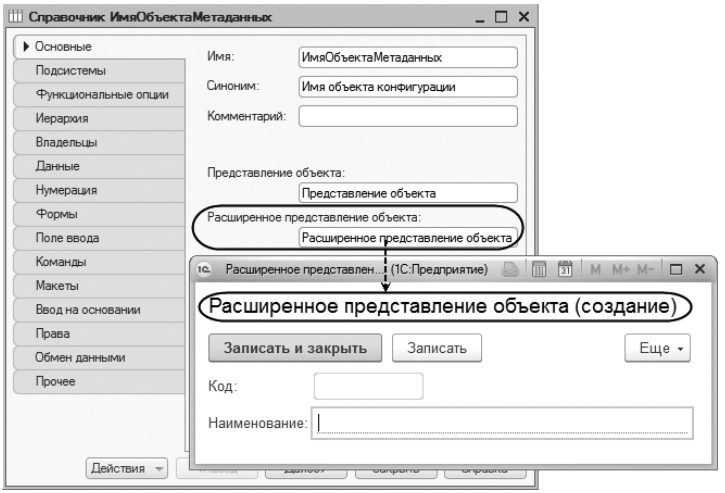


Рис. 2.116. Формирование заголовка формы по свойству «Расширенное представление объекта»

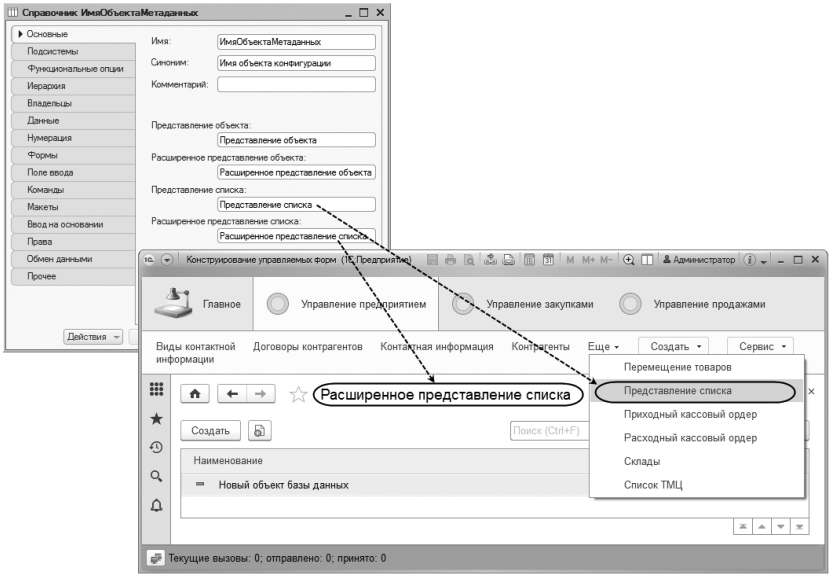


Рис. 2.117. Использование свойств объекта «Представление списка» и «Расширенное представление списка»

Если возникает необходимость задания собственного заголовка формы, то возможности автоматического формирования заголовка можно совмещать с текстом, введенным вручную. Для этого необходимо оставить флажок АвтоЗаголовок и применить свой Заголовок формы (рис. 2.118).

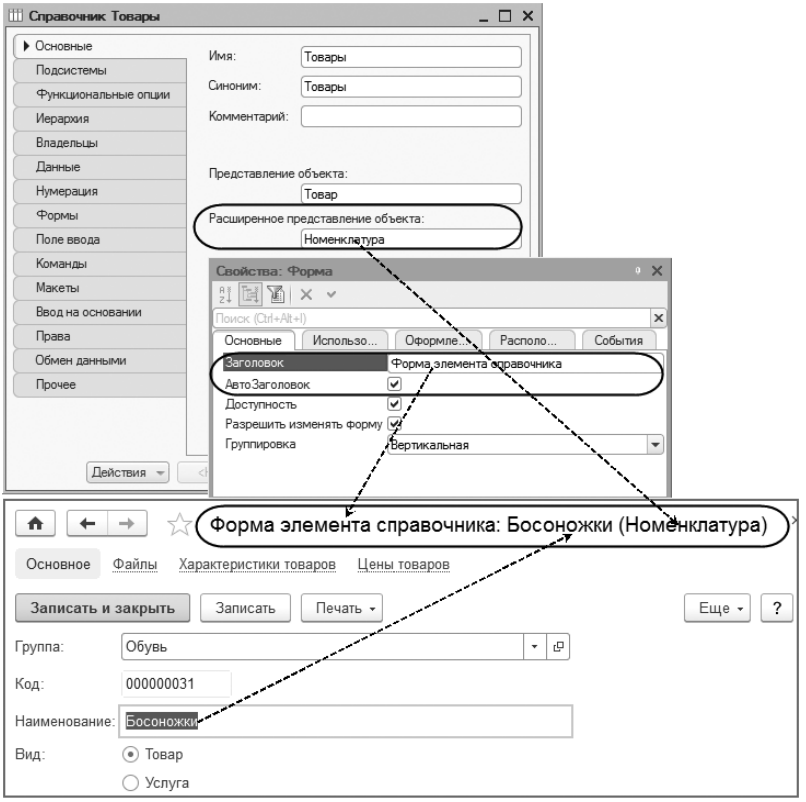


Рис. 2.118. Свойства «АвтоЗаголовок» и «Заголовок» формы

Если для формы необходимо применять только собственный заголовок, то свойство АвтоЗаголовок необходимо снять (рис. 2.119).

Немного по другим правилам формируются заголовки форм документа и задачи.

При создании нового документа (так же, как и для справочника) используется его расширенное представление (если оно задано).

После записи документа в базу данных заголовок формируется по правилу Представление объекта (если не задано, то Синоним) объекта конфигурации плюс представление ссылки на документ (рис. 2.120).

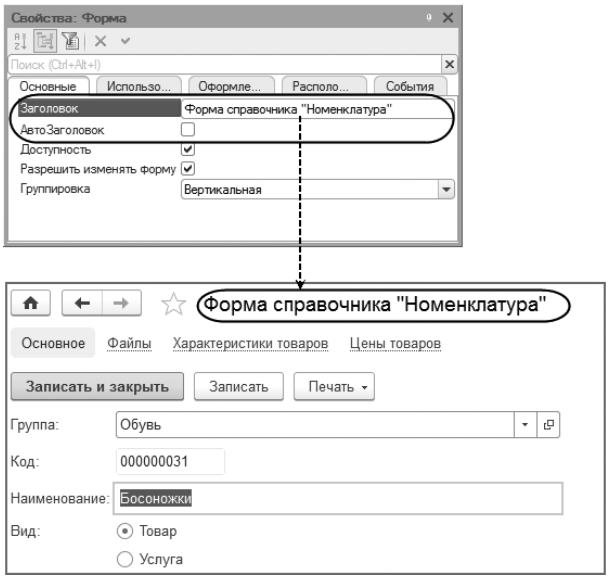


Рис. 2.119. Использование свойства формы «Заголовок»

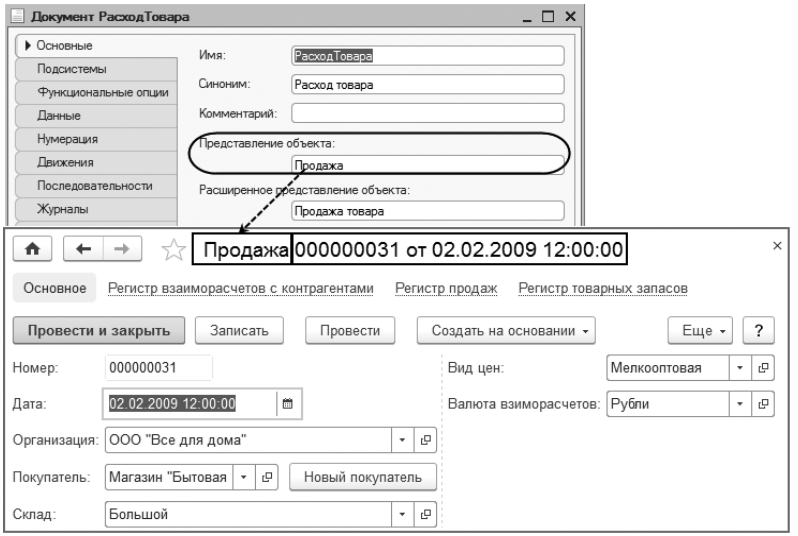


Рис. 2.120. Заголовок формы записанного документа

При формировании формы объекта конфигурации Задача система руководствуется наименованием задачи, ее номером, датой и одним из свойств объекта конфигурации, отвечающих за представление объекта (рис. 2.121).

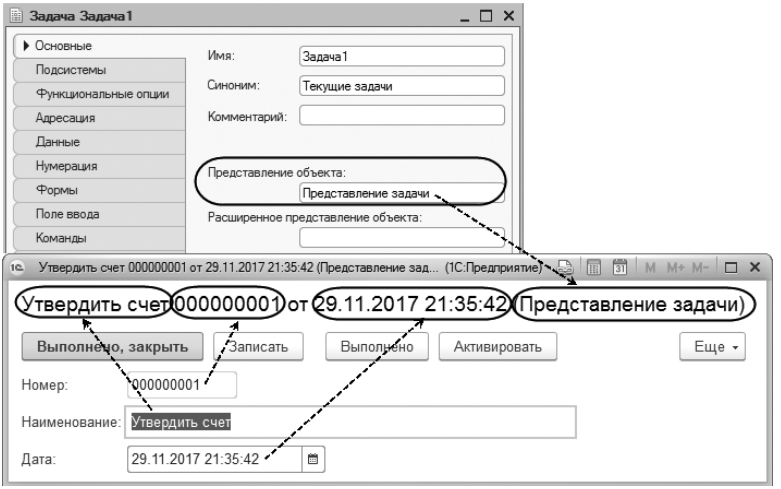


Рис. 2.121. Формирование заголовка формы задачи

ПРИМЕЧАНИЕ

Свойства объектов конфигурации, отвечающие за представление объектов и списков, влияют не только на формы, но и на интерфейс всего приложения. По каким правилам формируется интерфейс пользователя, рассказано в первой части этой книги в главе 1.9 «Настраиваем представление команд» на стр. 140.

Интерфейсные свойства реквизитов объектов конфигурации

Наиболее важные интерфейсные свойства существуют не только у элементов формы, но и у реквизитов объектов конфигурации. Это означает, что при установке такого свойства для реквизита оно автоматически действует на все формы, в которых данный реквизит отображается (при условии использования объекта в качестве основного реквизита формы). Если у элемента формы есть аналогичное

свойство, то оно позволяет переопределить для данной конкретной формы действие свойства.

ВНИМАНИЕ!

Если свойство существует у реквизита объекта и у элемента формы, который отображает данный реквизит, то использовать рекомендуется свойство реквизита. Аналогичным свойством элемента формы необходимо пользоваться только для переопределения значения свойства.

Некоторые свойства существуют также и «на более высоком уровне» – у прикладных объектов конфигурации, соответствующих типу реквизита. Если свойство существует и у реквизита объекта, и у соответствующего ему прикладного объекта, то рекомендуется использовать именно свойство самого объекта. Но если нужно, то можно переопределить значение свойства у конкретного реквизита, имеющего тип этого объекта конфигурации.

Более подробно эта «схема» рассматривается в разделе «Быстрый выбор» на стр. 336.

Большинство свойств реквизита, которые влияют на поведение элемента формы, находятся в разделе Представление его свойств или свойств самого прикладного объекта (рис. 2.122).

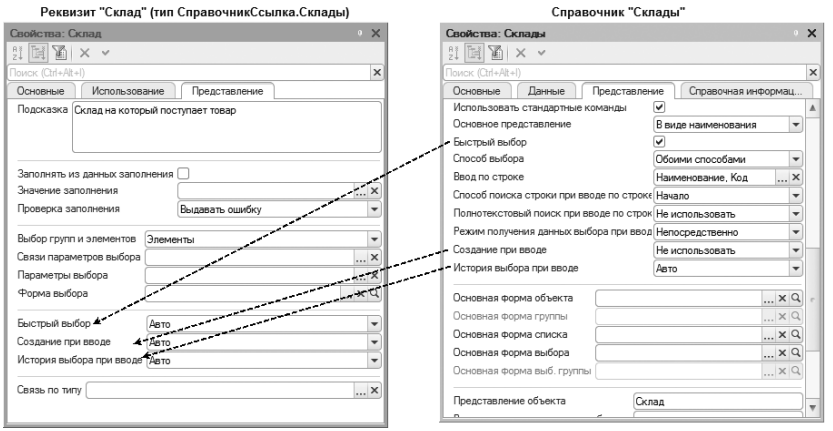


Рис. 2.122. Раздел «Представление» свойств реквизитов и прикладных объектов

Выбор групп и элементов

В случае организации выбора в элементе формы данных иерархических объектов базы данных (групп, элементов), например справочников, выборку можно ограничить с помощью настройки свойства реквизита Выбор групп и элементов. Таким образом, если по логике работы пользователю необходимо предоставить для выбора только группы справочника, то, установив свойство в значение Группы, разработчик ограничит варианты выбора данных информационной базы.

Быстрый выбор

Часто для повышения производительности действий пользователей и удобства их работы выбор элементов удобно делать не в отдельном окне формы выбора, а в выпадающем списке. Реализовать подобную настройку выбора элементов можно с помощью установки свойства БыстрыйВыбор.

За способ выбора объектов при заполнении полей ввода отвечают свойства прикладных объектов СпособВыбора и БыстрыйВыбор. Свойство БыстрыйВыбор отвечает за режим выбора по умолчанию. Свойство СпособВыбора может принимать значения Обоими способами, Из формы и Быстрый выбор.

Допустим, в конфигурации есть справочник Склады. У него свойство СпособВыбора равно Обоими способами, а также установлено свойство БыстрыйВыбор. В этом случае по умолчанию выбор из значений справочника Склады во всем прикладном решении будет осуществляться в режиме быстрого выбора (рис. 2.123).

Если требуется переопределить режим выбора для реквизита в каком-то конкретном месте прикладного решения, нужно установить у него свойство БыстрыйВыбор в значение Использовать или Не использовать. При этом значение свойства СпособВыбора прикладного объекта не должно ему противоречить, т. к. оно является определяющим.

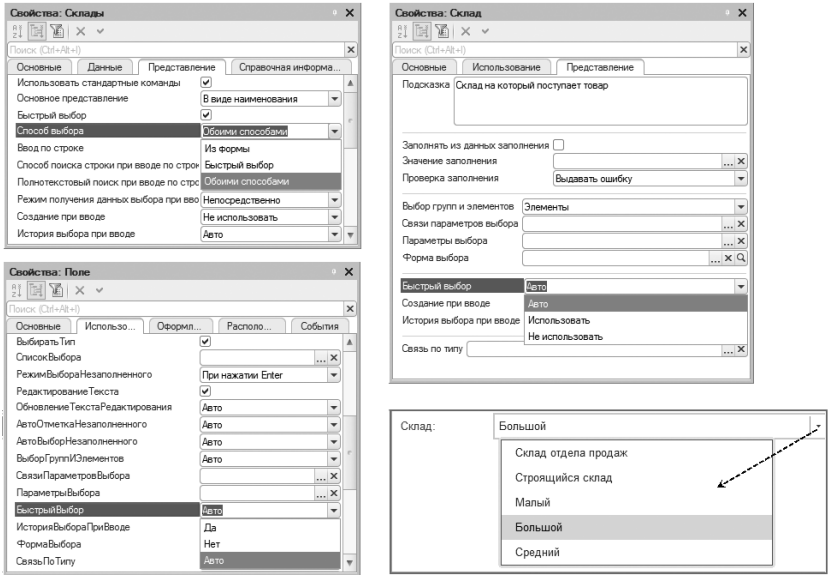


Рис. 2.123. Быстрый выбор значений

То есть если нужно запретить быстрый выбор для реквизита, то нужно установить у него свойство БыстрыйВыбор в значение Не использовать. При этом свойство СпособВыбора прикладного объекта должно принимать значения Обоими способами или Из формы. Если же нужно разрешить быстрый выбор для реквизита, то нужно установить у него свойство БыстрыйВыбор в значение Использовать. При этом свойство СпособВыбора прикладного объекта должно принимать значения Обоими способами или Быстрый выбор.

Свойство БыстрыйВыбор существует также и у элемента формы, связанного с реквизитом. О нем рассказывалось на стр. 336.

Таким образом, получается трехуровневая схема управления режимом выбора:

1. Сначала платформа анализирует свойство БыстрыйВыбор элемента формы.
2. Если значение свойства равно Авто, то оно анализируется у реквизита объекта.

3. Если значение свойства равно Авто, выполняется анализ свойств СпособВыбора и БыстрыйВыбор прикладного объекта, соответствующего типу реквизита. Если на каком-то из первых двух уровней (элемент формы и реквизит объекта) значение свойства БыстрыйВыбор отлично от Авто, анализ прекращается и выполняется выбор в определенном режиме.

История выбора при вводе

Свойство ИсторияВыбораПриВводе позволяет сохранять и отображать историю выбора при вводе ссылочных значений в поле формы. История выбора состоит из наиболее часто используемых и последних выбранных значений данного ссылочного типа. Свойство ИсторияВыбораПриВводе может принимать значения Авто и Не использовать.

Управление историей выбора реализовано с помощью трехуровневой схемы:

1. Сначала платформа анализирует свойство ИсторияВыбораПриВводе для поля ввода формы.
2. Если значение свойства равно Авто, то выполняется анализ свойства ИсторияВыбораПриВводе у реквизита, отображаемого полем ввода.
3. Если значение свойства равно Авто, выполняется анализ свойства ИсторияВыбораПриВводе у объекта конфигурации, соответствующего типу реквизита.

Если на всех «уровнях» принятия решения свойство установлено в значение Авто, то история выбора при вводе в ссылочное поле будет сохраняться и отображаться в том случае, если это поле не находится в режиме выбора из списка или в режиме быстрого выбора (рис. 2.124).

Иногда требуется отключить историю выбора ссылочных значений в каком-то конкретном месте прикладного решения. В этом случае нужно запретить показ истории выбора при вводе на уровне реквизита объекта конфигурации. Для этого нужно установить у него свойство ИсторияВыбораПриВводе в значение Не использовать.

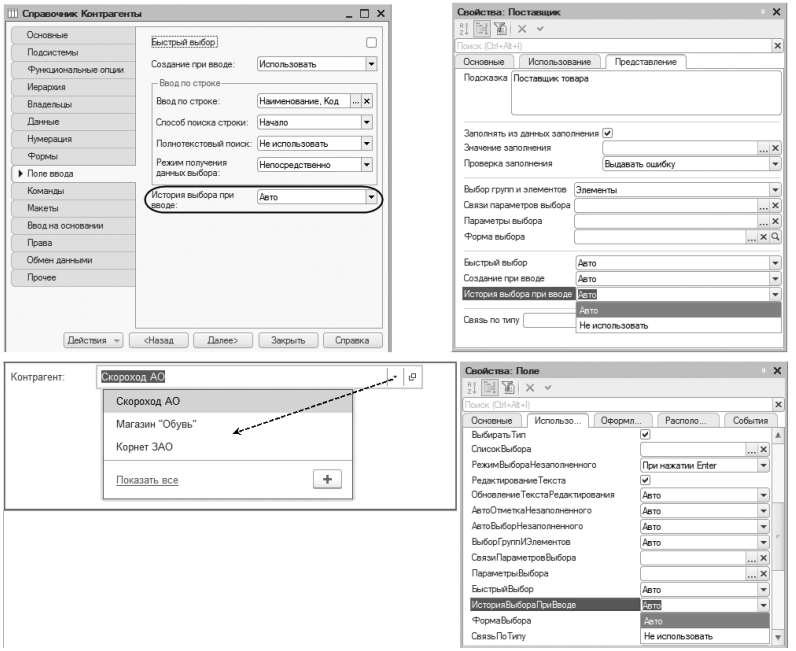


Рис. 2.124. История выбора при вводе ссылочных значений

Обычно историю выбора рекомендуется отключать в свойствах объекта конфигурации, если ее использование не соответствует прикладной логике. То есть когда требуется выбирать все время новые значения, которые еще не использовались ранее, – например, когда необходимо выбирать и закрывать заказы покупателей.

Затем, чтобы пользователю перед началом выбора в поле ввода не отображалось окно со ссылкой Показать все для вызова формы выбора, необходимо во всех ссылающихся на этот реквизит полях ввода установить для следующих свойств указанные ниже значения:

- КнопкаВыпадающегоСписка – Нет;
- КнопкаВыбора – Да;
- ОтображениеКнопкиВыбора – В поле ввода.

На рис. 2.125 внизу показан правильный вариант, при котором кнопка выбора отображается в поле ввода вместо кнопки

выпадающего списка, а при нажатии на кнопку выбора сразу вызывается форма выбора объекта конфигурации.

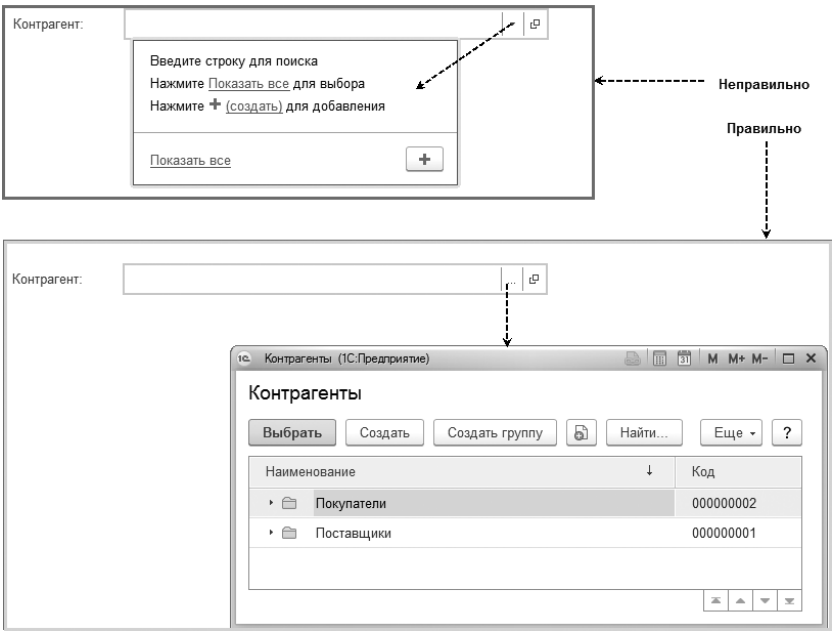


Рис. 2.125. Отключение истории выбора при вводе

ПРИМЕЧАНИЕ

О свойстве ИсторияВыбораПриВводе у элемента формы рассказано на стр. 338.

Связи параметров выбора

В ряде задач, решаемых пользователями с помощью элементов формы, необходимо ограничивать количество предлагаемых для выбора данных. Так, например, в документе оплаты поставщику товаров, при выборе расчетного счета поставщика желательно видеть не все расчетные счета контрагентов, которые есть в соответствующем справочнике, а только расчетные счета того поставщика, которому совершается оплата. Сделать это можно с помощью настройки свойства СвязиПараметровВыбора.

Например, на показанном ниже рисунке для реквизита РасчетныйСчетПоставщика документа Оплата свойство СвязиПараметровВыбора установлено в значение Поставщик, которое будет устанавливаться в поле отбора Владелец (рис. 2.126).

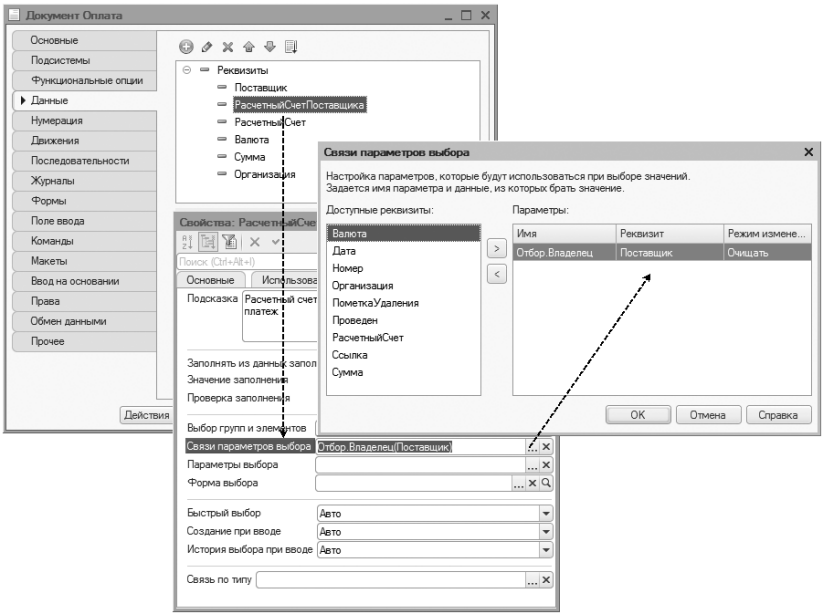


Рис. 2.126. Установка свойства «Связи параметров выбора»

Этот отбор будет применяться при открытии формы выбора, при отображении списка быстрого выбора и при выполнении ввода по строке (рис. 2.127).

Пример использования свойства рассматривается в главе 2.9 на стр. 381.

Таким образом, при установке связей параметров выбора значение одного реквизита объекта конфигурации (в нашем случае реквизита Поставщик документа Оплата), накладывает ограничение на выбор значений другого реквизита (РасчетныйСчетПоставщика) этого же объекта.

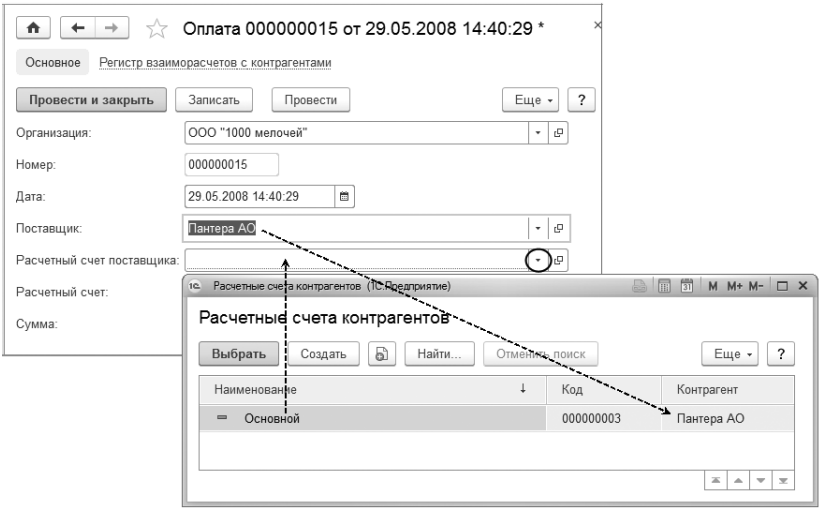


Рис. 2.127. Использование свойства «Связи параметров выбора»

ПРИМЕЧАНИЕ

О свойстве СвязиПараметровВыбора у элемента формы рассказано на стр. 340.

Параметры выбора

Свойство ПараметрыВыбора позволяет указать значения параметров, которые будут применяться при выборе значения реквизита. Но этот выбор будет выполняться не на основании значений других реквизитов формы, а на основании условий, наложенных на свойства самого выбираемого объекта. Значениями отбора могут выступать ссылки на значения справочников, перечислений, планов видов характеристик, значения примитивных типов данных и список значений фиксированного массива.

Например, при учете поступления товаров (документ ПриходТовара, реквизит табличной части Товар) необходимо запретить выбор услуг, т. е. ограничить выбор только теми товарами, у которых реквизит Вид равен значению `Перечисление.ВидыТоваров.Товар` (рис. 2.128).

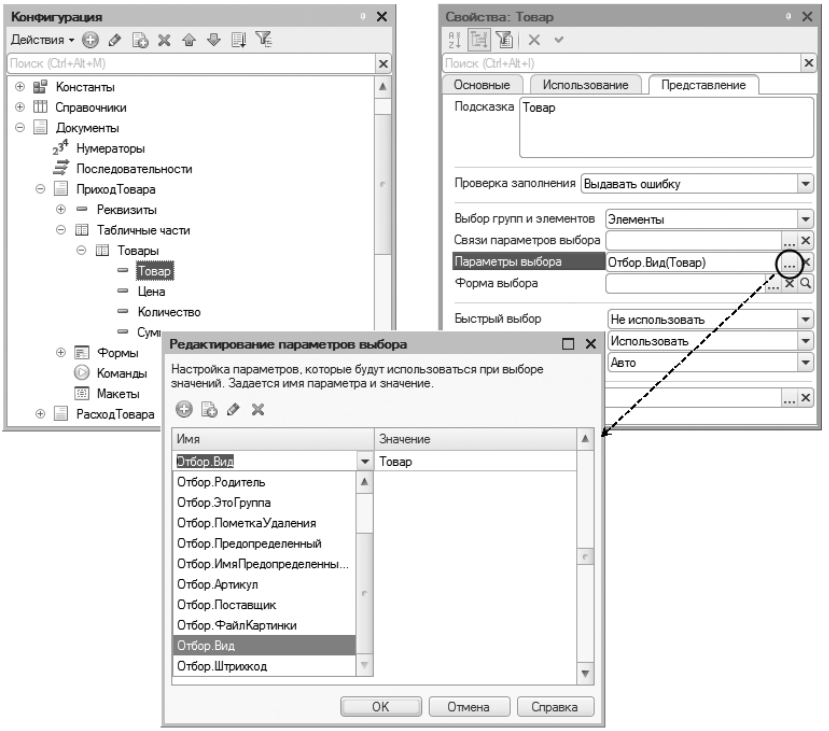


Рис. 2.128. Установка свойства «Параметры выбора»

Этот отбор будет применяться при открытии формы выбора, при отображении списка быстрого выбора и при выполнении ввода по строке (рис. 2.129).

Если имя у какого-либо параметра свойства СвязиПараметровВыбора совпадает с именем какого-либо параметра свойства ПараметрыВыбора, то приоритет отдается элементу из свойства СвязиПараметровВыбора (в том случае, если значение поля, указанного в параметре Реквизит, заполнено).

ПРИМЕЧАНИЕ

О свойстве ПараметрыВыбора у элемента формы рассказано на стр. 342.

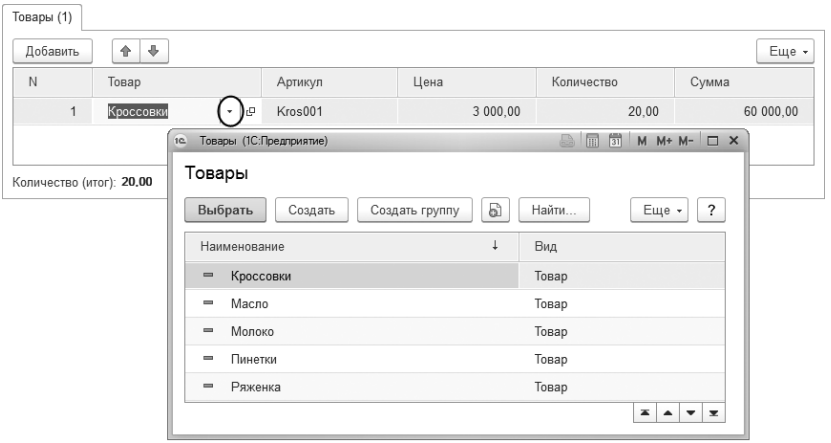


Рис. 2.129. Использование свойства «Параметры выбора»

Создание при вводе

Свойство СозданиеПриВводе определяет наличие кнопки для создания нового элемента в выпадающем списке, открывающемся под ссылочным полем.

Допустим, в конфигурации есть справочник Товары. У него свойство СозданиеПриВводе установлено в значение Использовать. В этом случае по умолчанию во всем прикладном решении при выборе товаров в выпадающем списке, открывающемся под соответствующим полем, будет присутствовать кнопка создания нового элемента справочника Товары (рис. 2.130).

При наличии кнопки создания в выпадающем списке удобно сразу начинать ввод нового элемента справочника в поле ввода. Если совпадений с новым значением не найдено, можно нажать кнопку создания нового элемента (с пиктограммой «+») и продолжить ввод наименования в специальной форме. При этом введенные ранее символы уже будут подставлены в наименование нового элемента справочника (рис. 2.131).

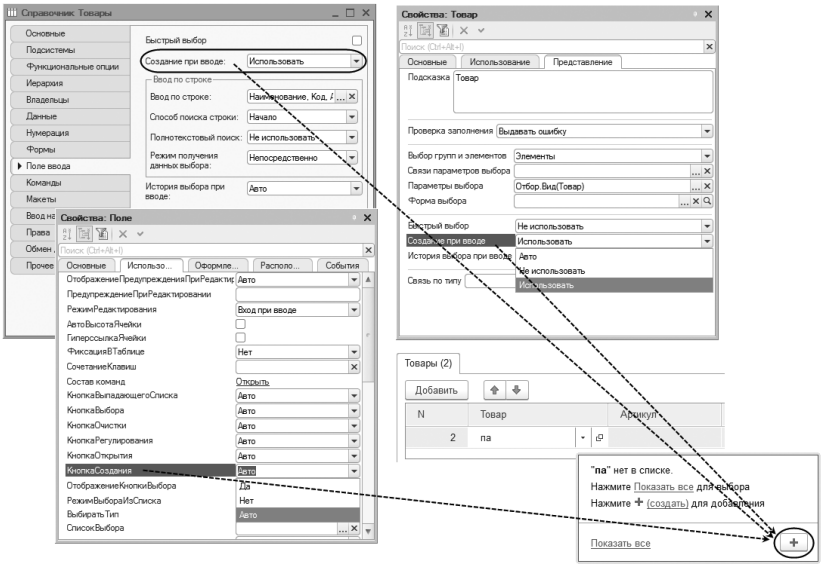


Рис. 2.130. Свойство «Создание при вводе»

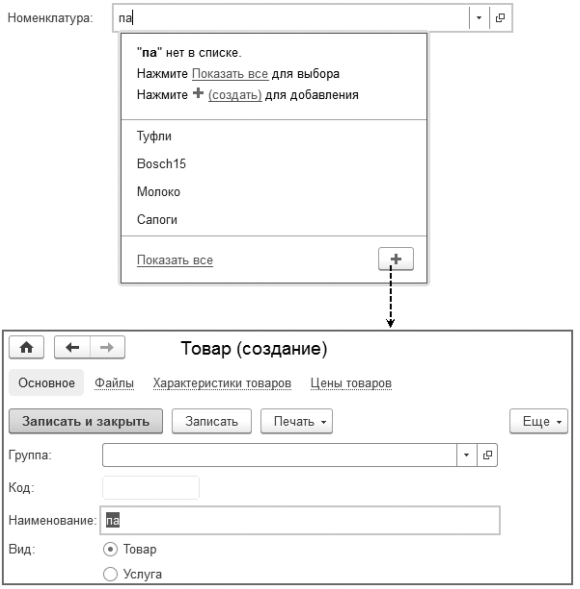


Рис. 2.131. Создание нового товара

После записи нового элемента ссылка на этот элемент автоматически подставится в поле ввода.

Если требуется переопределить режим создания при вводе для реквизита в каком-то конкретном месте прикладного решения, нужно явно установить у него свойство СозданиеПриВводе в значение Использовать или Не использовать.

При создании элемента из выпадающего списка, связанного с полем ввода, существует возможность автоматической и программной проверки созданного элемента на соответствие параметрам выбора и связям параметров выбора, которые использовались при открытии формы нового элемента. Если элемент не соответствует параметрам выбора, будет выдано диагностическое сообщение и созданный элемент не будет помещен в поле ввода.

Подробнее о программной проверке элемента, созданного при вводе в ссылочное поле, рассказано в третьей части, в разделе «Поле ввода», на стр. 667.

О свойстве КнопкаСоздания у элемента формы рассказано на стр. 313.

Связь по типу

Свойство СвязьПоТипу позволяет указать реквизит объекта, который будет поставлять тип значения для текущего реквизита. Указание такой связи используется в том случае, если необходимо, например, дать возможность пользователям самим определять дополнительные свойства тех или иных объектов базы данных. Настройка связи по типу имеет смысл для реквизитов с составным типом данных, логически связанных с другим реквизитом.

Например, в конфигурации есть план видов характеристик, который хранит виды характеристик товаров. Значения характеристик могут быть различного типа (Число, Дата, Булево, СправочникСсылка.<Имя> и др.). Значения характеристик хранятся в регистре сведений, измерением которого является вид характеристики.

Для ресурса Значение регистра сведений ХарактеристикиТоваров устанавливается связь по типу с измерением ВидХарактеристики. В результате тип значения выбранной характеристики определяется

типом, который задан для нее в плане видов характеристик (рис. 2.132).

О свойстве СвязьПоТипу у элемента формы рассказано на стр. 346.

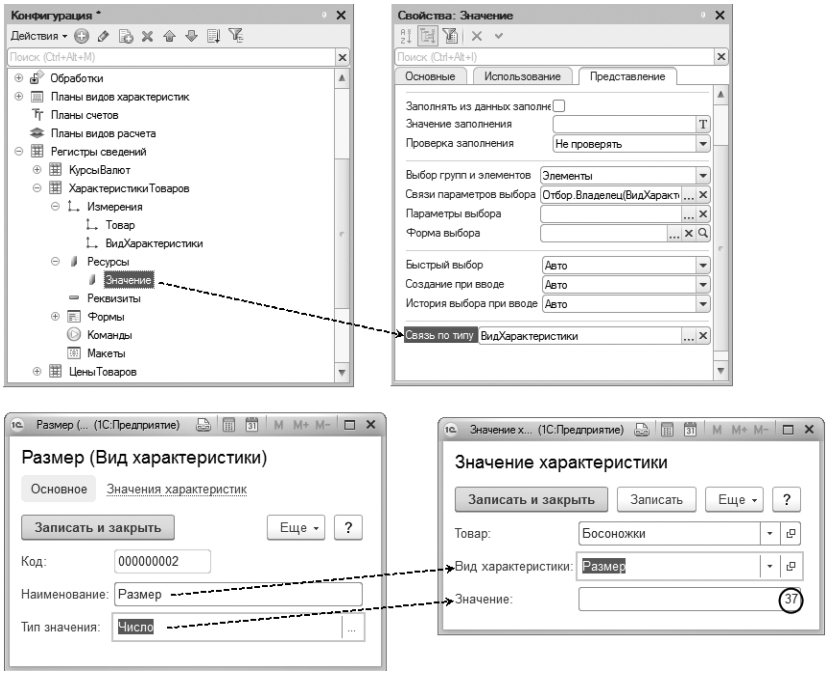


Рис. 2.132. Использование свойства «Связь по типу»

Форма выбора

В свойстве ФормаВыбора можно указать, какая из форм объекта будет использоваться в качестве основной формы выбора значения реквизита. Используется для реквизитов, тип которых образован объектом конфигурации, имеющим подчиненные формы. Стандартно для выбора ссылочного реквизита используется форма, назначенная в качестве основной формы выбора для объекта конфигурации, на который он ссылается. Но с помощью свойства ФормаВыбора можно установить для реквизита другую форму, которая будет вызываться при выборе значений в связанном с ним поле.

Проверка заполнения

Свойство ПроверкаЗаполнения отвечает за автоматическую проверку заполнения реквизита (обязательный реквизит). При установке свойства реквизита в значение Выдавать ошибку у связанного с ним элемента формы изменится визуальное представление – элемент будет подчеркнут красной линией. В случае необходимости переопределения свойства нужно будет воспользоваться обработчиком события ПроверкаЗаполнения прикладного объекта, на который ссылается данный реквизит. Пример использования данного свойства рассматривается в главе 2.8 на стр. 377.

Значение заполнения

Свойство ЗначениеЗаполнения позволяет указать начальное значение реквизита. Для выбора доступны значения, созданные разработчиком в режиме Конфигуратор (значения перечислений, предопределенные элементы).

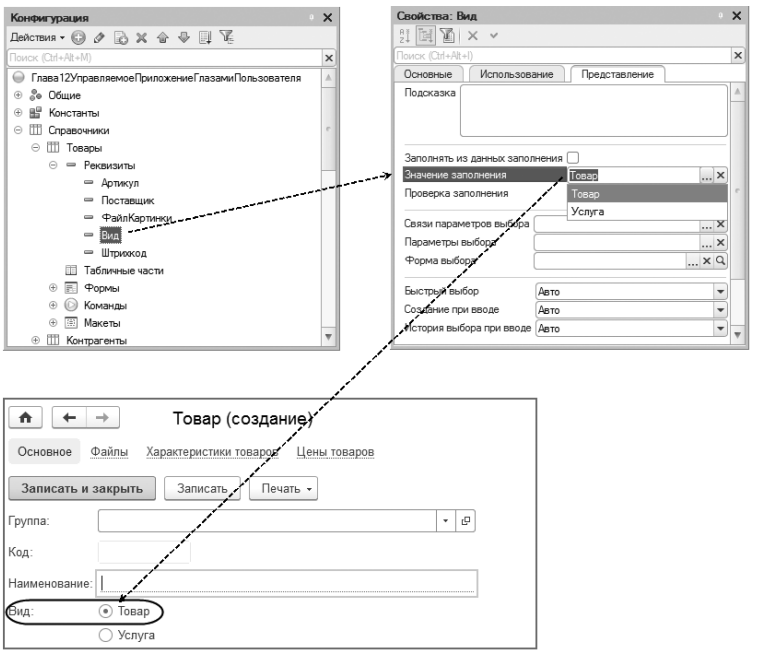


Рис. 2.133. Свойство «Значение заполнения»

Примером использования свойства может служить установка вида товара при создании нового элемента справочника. По умолчанию реквизит устанавливается в значение Товар перечисления ВидыТоваров (рис. 2.133).

Заполнять из данных заполнения

Не менее интересным является свойство ЗаполнятьИзДанныхЗаполнения. При определенных обстоятельствах данный флажок может отменять значение свойства ЗначениеЗаполнения. Для некоторых реквизитов флажок Заполнять из данных заполнения устанавливается автоматически. Если разработчик заинтересован в том, чтобы его данные из свойства ЗначениеЗаполнения использовались всегда, флажок Заполнять из данных заполнения должен быть снят.

Если флажок Заполнять из данных заполнения установлен и происходит интерактивное создание нового объекта, то реквизит будет заполняться с учетом установленного отбора. Пример такого заполнения рассматривается в главе 2.9 на стр. 381.

При интерактивном вводе на основании в структуру данных заполнения формы передается ссылка на объект-основание, и реквизиты создаваемого объекта заполняются исходя из значений реквизитов основания.

Программно объект можно заполнить, вызвав его метод Заполнить(). У метода существует параметр ДанныеЗаполнения, исходя из которого и происходит заполнение реквизитов объекта.

Еще одним из способов заполнения реквизитов формы первоначальными данными является вызов глобального метода ОткрытьФорму(). Если в метод передать параметр формы ЗначенияЗаполнения, то форма будет открываться с учетом переданных в параметре значений.

Стандартные реквизиты объектов конфигурации

У каждого из классов объектов конфигурации существует свой набор стандартных реквизитов. Например, у справочников это Ссылка, Код, Наименование, ПометкаУдаления и т. п.; у документов: Номер, Дата, ПометкаУдаления, Ссылка, Проведен.

Чтобы открыть и изменить свойства стандартных реквизитов объекта конфигурации, нужно выполнить команду Стандартные реквизиты из его контекстного меню. Или же можно нажать кнопку Стандартные реквизиты на закладке Данные окна редактирования свойств объекта конфигурации (рис. 2.134).

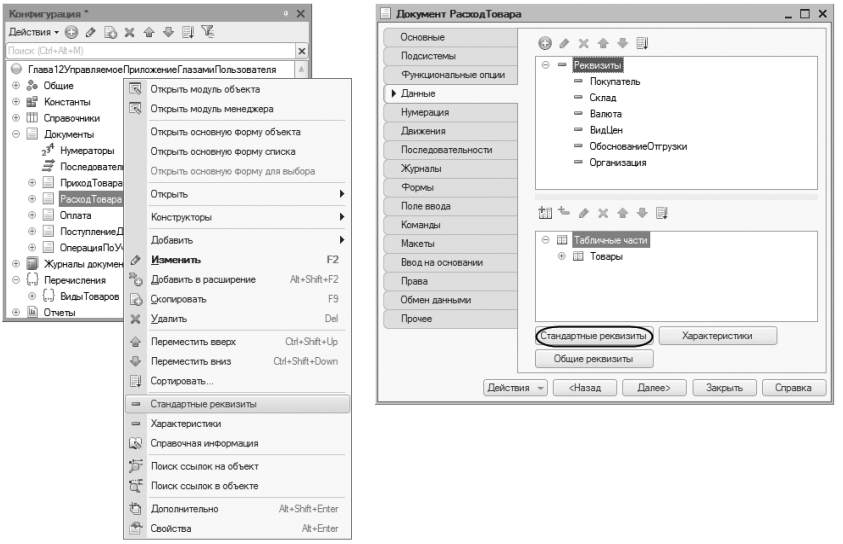


Рис. 2.134. Открытие списка стандартных реквизитов

Состав свойств каждого из этих реквизитов различается и зависит от типа данных реквизита. Большинство свойств стандартных реквизитов совпадают со свойствами реквизитов объектов и свойствами элементов формы, и они нами уже рассмотрены.

Для стандартных реквизитов объектов конфигурации, автоматически поставляемых платформой, так же как и для «обычных» реквизитов, создаваемых разработчиком, можно изменять их свойства, влияющие на интерфейс приложения.

Например, для реквизита Родитель иерархического справочника Товары можно задать значение свойства Синоним как «Группа». В результате в режиме 1С:Предприятие во всех формах, где используется реквизит Родитель, вместо непонятного для пользователя заголовка «Родитель» он будет иметь заголовок «Группа» (рис. 2.135).

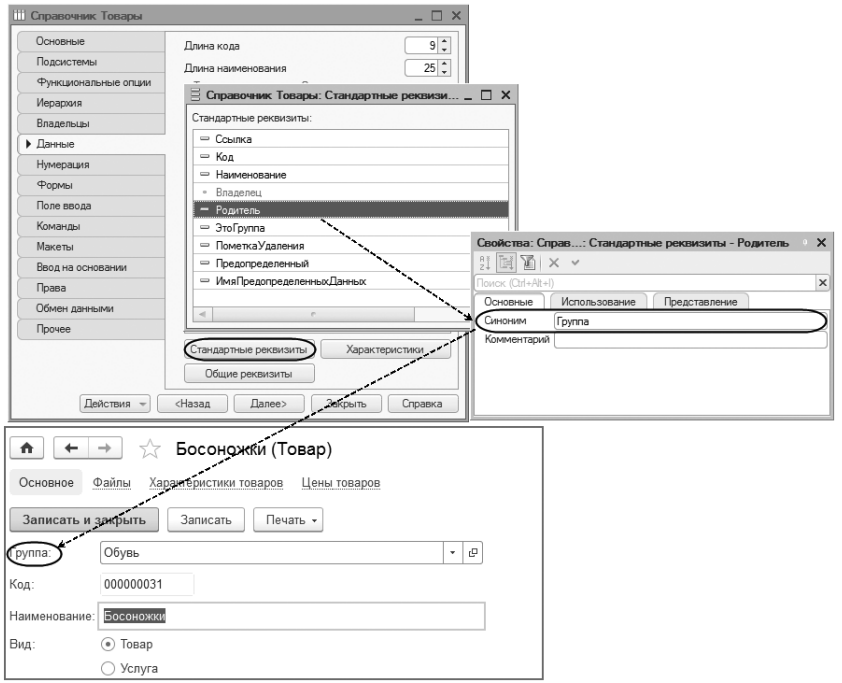


Рис. 2.135. Синоним стандартного реквизита

Глава 2.5. Реквизиты формы

Достаточно часто при разработке прикладных решений перед разработчиками стоит задача размещения на форме различных данных, не связанных с каким-либо определенным объектом. Это могут быть данные подчиненных справочников, регистров, таблицы, полученные в результате работы процедур встроенного языка.

ВНИМАНИЕ!

В формах все данные, предназначенные для изменения, должны быть описаны в редакторе формы в виде реквизитов.

Во многих главах книги уже неоднократно упоминался *основной реквизит формы*. Такой реквизит автоматически добавляется конструктором форм, если форма создается подчиненной объекту конфигурации и выбирается вид формы этого объекта, а не произвольная форма. По умолчанию у такой формы после ее создания есть только один реквизит, который и является основным. В окне реквизитов формы он выделен жирным шрифтом (рис. 2.136).

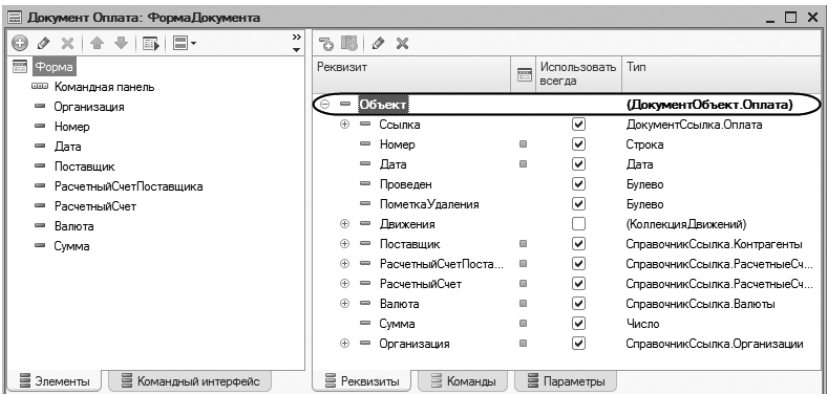


Рис. 2.136. Основной реквизит формы

Флажок Использовать всегда позволяет влиять на передачу данных формы в процессе ее работы между клиентской частью и серверной.

Флажок оказывает влияние на внутреннее наполнение формы и не влияет на пользовательское представление.

По умолчанию платформа устанавливает этот флажок для всех данных, отображаемых в форме. И снимать его не нужно. Но если, например, какая-то колонка динамического списка не отображается в форме, но используется в коде, то будет получена ошибка. Потому что в этом случае данные колонки не будут переданы с сервера на клиент. Чтобы избежать ошибки, нужно установить флажок Использовать всегда у тех данных, которые не отображаются в интерфейсе, но к которым обращаются из встроенного языка.

Добавление нового реквизита формы выполняется стандартным способом – с помощью кнопки командной панели в окне реквизитов редактора формы. Реквизиты формы обладают набором свойств (см. рис. 2.137), позволяющих влиять на их поведение.

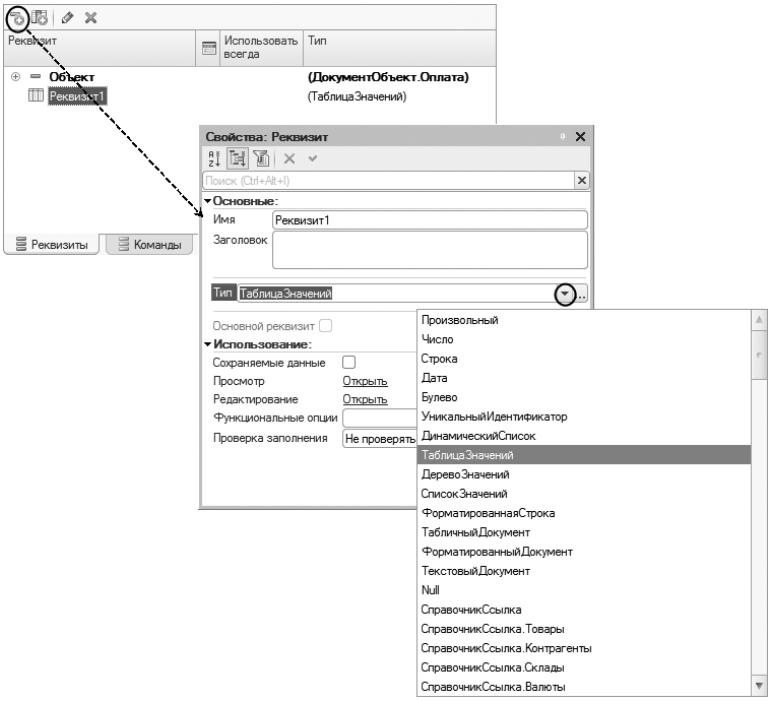


Рис. 2.137. Установка свойства «Тип» реквизита формы

Функциональные и интерфейсные свойства реквизитов формы (ФункциональныеОпции, Просмотр, Редактирование) рассматриваются в главе 2.7 на стр. 361. Со свойством ПроверкаЗаполнения можно познакомиться в главе 2.8 на стр. 377.

Важнейшим свойством, которое определяет не только то, что за данные будет поставлять форме реквизит, но и за то, как будет выглядеть элемент формы, отображающий эти данные, является Тип (рис. 2.137).

Пример добавления реквизитов формы рассматривается в главе 2.9 на стр. 381.

Глава 2.6. Командный интерфейс окна клиентского приложения

Панель навигации

Командный интерфейс окна клиентского приложения формируется системой частично автоматически, на основании анализа объектов конфигурации. При этом разработчик имеет возможность добавить в него какие-то команды или, наоборот, скрыть. А также настроить их ролевую видимость.

Так, например, после добавления в дерево конфигурации подчиненного справочника или регистра сведений, у которого тип значения измерения является ссылкой на текущий справочник и при этом установлен признак свойства измерения Ведущее (рис. 2.138), в форме текущего справочника системой автоматически будут сформированы команды для открытия форм объектов конфигурации, содержащих зависимые данные.

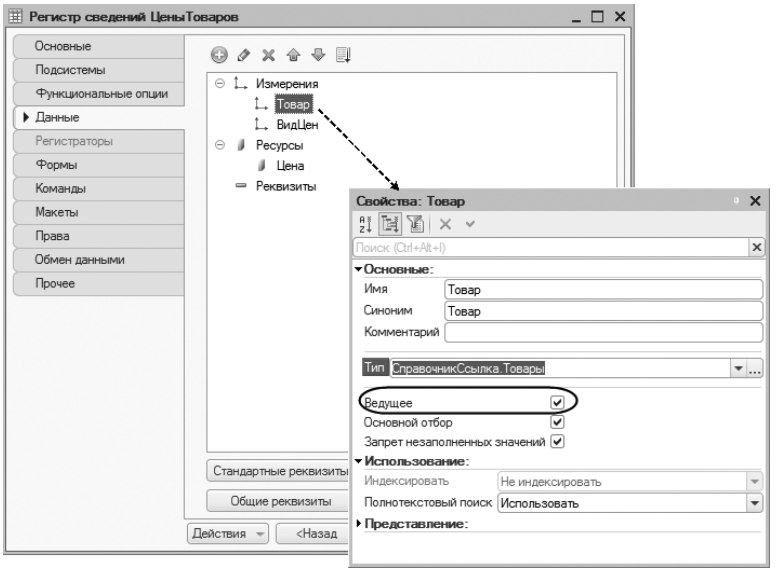


Рис. 2.138. Свойство измерения «Ведущее»

Такие команды будут доступны в панели навигации окна клиентского приложения (рис. 2.139).

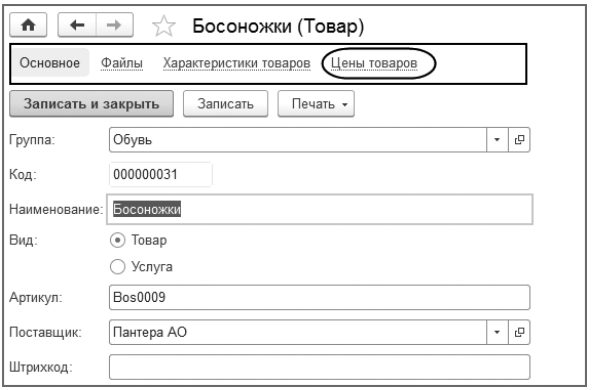


Рис. 2.139. Команды панели навигации окна клиентского приложения

В режиме разработки формы такие команды можно увидеть и отредактировать в редакторе формы. Они будут находиться на закладке Командный интерфейс (рис. 2.140). При этом у разработчика формы есть возможность повлиять на видимость таких команд в пользовательском режиме работы.

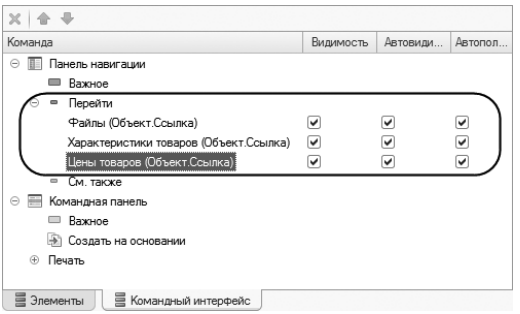


Рис. 2.140. Редактирование командного интерфейса формы

После этого платформа в пользовательском режиме может ограничить состав команд, заданный в конфигураторе, в зависимости от прав доступа пользователей.

Командная панель основной формы

Если на основании какого-либо объекта информационной базы можно вводить новые данные (например, на основании элемента справочника Контрагенты можно ввести документ РаходТовара (Продажа), см. рис. 2.141), то в форме элемента-основания системой формируется предопределенное подменю Создать на основании.

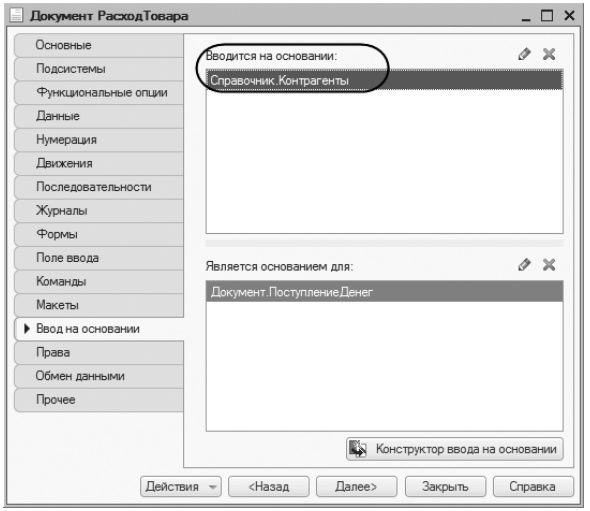


Рис. 2.141. Объект-основание для текущего объекта

В пользовательском режиме работы такое подменю формируется в командной панели основной формы окна клиентского приложения (рис. 2.142). Такой способ ввода новых объектов базы данных позволяет значительно повысить производительность работы пользователей, уменьшить количество допущенных в работе ошибок, избавить от постоянного нудного и однообразного ввода одних и тех же данных.

Управлять видимостью подменю Создать на основании, а также его составом можно в процессе разработки формы на закладке Командный интерфейс (рис. 2.143).

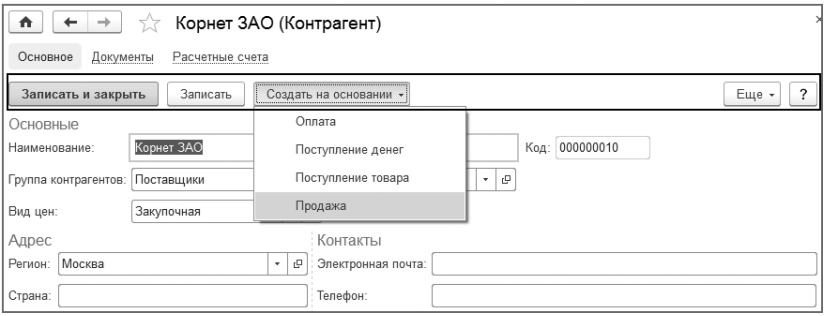


Рис. 2.142. Подменю «Создать на основании»

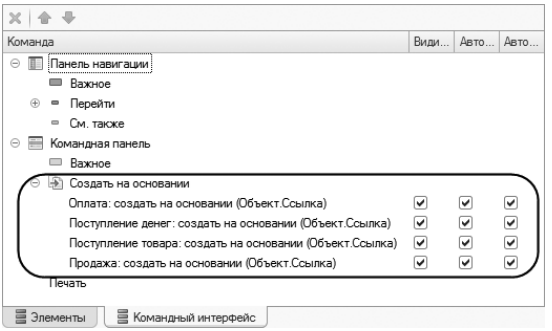


Рис. 2.143. Управление видимостью команд командной панели формы

Команды формы

Помимо заполнения каких-либо элементов формы пользователю достаточно часто приходится командовать форме, что сделать в том или ином случае. Все команды, которые могут быть назначены кнопкам и использованы в форме, сгруппированы в окне команд (рис. 2.144).

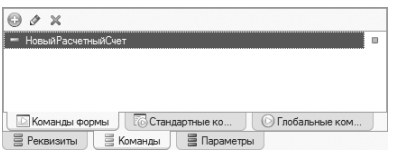


Рис. 2.144. Команды формы

При этом для удобства разработчика все команды разделены на три категории:

- Команды формы. Именно в данном окне с помощью кнопки Добавить можно создавать новые команды формы. При этом все создаваемые команды являются «программируемыми». Описание алгоритма работы команд выполняется разработчиком в модуле формы, с помощью встроенного языка.
- Стандартные команды. В данном разделе собраны все команды, которые предоставляются основным реквизитом формы. Если в форме есть элементы, отображающие списки (наборы записей, динамические списки, табличные части объектов и так далее), а также табличные элементы, форматированные элементы, графические схемы или планировщики, то команды таких элементов также присутствуют в данной категории команд (рис. 2.145).

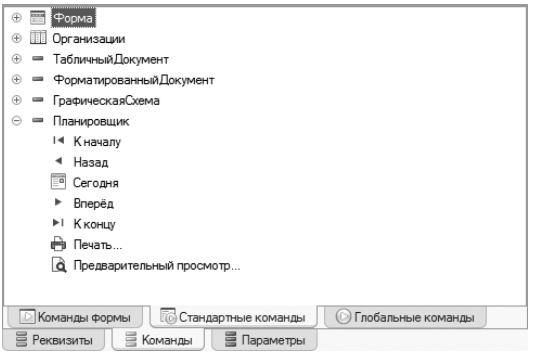


Рис. 2.145. Стандартные команды формы

- Глобальные команды. Команды этой группы могут являться параметризованными и непараметризованными. Это команды глобального командного интерфейса. Параметризованные команды отображаются в форме, только если для них в форме существует источник параметров с соответствующим типом данных (рис. 2.146).

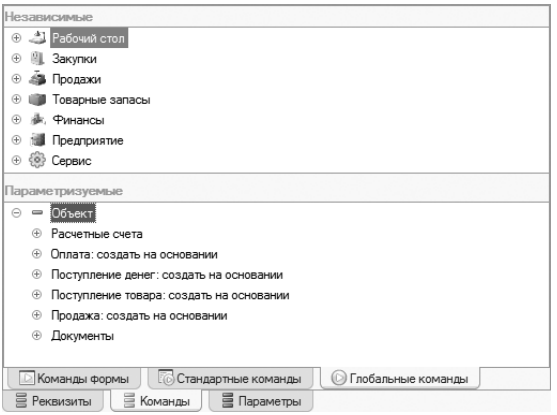


Рис. 2.146. Глобальные команды формы

Глава 2.7. Управление видимостью элементов формы

Влияние прав и ролей пользователя на элементы формы

Для ограничения доступа к данным информационной базы в пользовательском режиме работы в среде «1С:Предприятия» применяются объекты конфигурации Роль. Каждая роль, присутствующая в том или ином прикладном решении, описывает набор прав доступа к объектам информационной базы. Так как работа пользователя с объектами информационной базы происходит в формах, права доступа к таким объектам законным образом влияют на внешний вид формы.

Права доступа и роли пользователей оказывают влияние на все элементы формы, на реквизиты формы, на состав ее команд и системных командных панелей. Кроме того, оказывается косвенное влияние через объекты конфигурации и их реквизиты.

Однако ограничение доступа к данным и удобство пользования теми данными, к которым имеется доступ, – это два разных вопроса. И для решения второго вопроса существует ряд свойств, помогающих разработчику настроить доступ к разрешенным данным удобным для пользователя образом. Эти свойства носят вполне понятные названия и не требуют уточнения.

Например, для элементов формы это свойство Пользовательская видимость (рис. 2.147); для реквизитов формы доступны два свойства: Просмотр и Редактирование (рис. 2.148); для команд формы это Использование (рис. 2.149); для командного интерфейса формы – Видимость (рис. 2.150).

Как видно из приведенных рисунков, разработчик может либо полностью запретить видимость (просмотр, редактирование) того или иного элемента формы (реквизита, команды), либо применить запрет для конкретной роли.

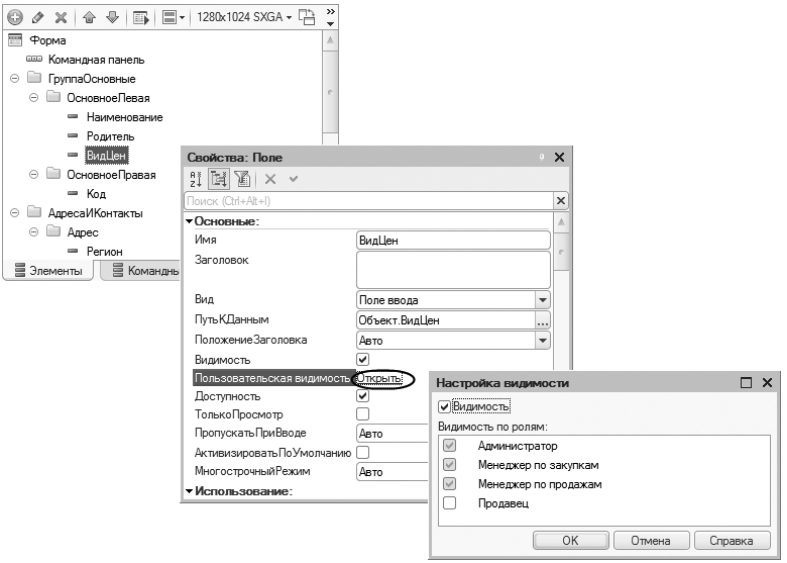


Рис. 2.147. Свойство «Пользовательская видимость» элементов формы

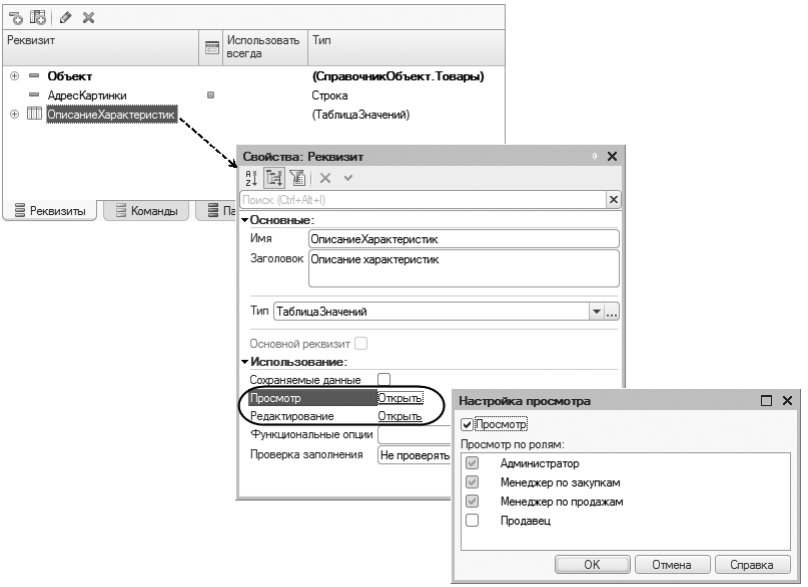


Рис. 2.148. Свойства «Просмотр» и «Редактирование» реквизита формы

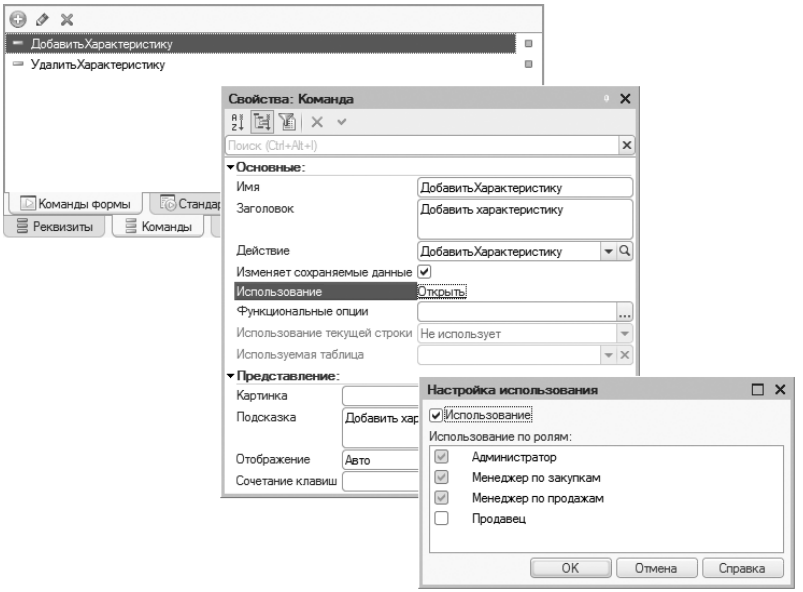


Рис. 2.149. Свойство «Использование» команды формы

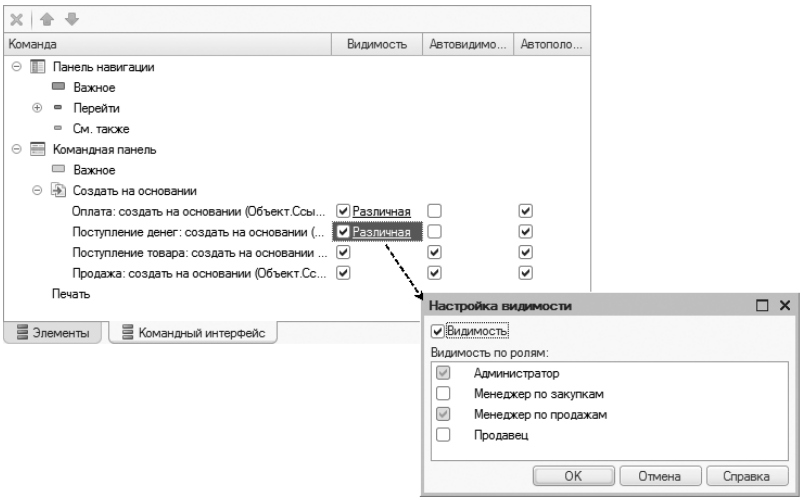


Рис. 2.150. Свойство «Видимость» командного интерфейса формы

Для элементов формы полное выключение свойства Видимость у родительского элемента, например Группы, означает исключение из пользовательского интерфейса формы всех дочерних элементов.

Рассмотрим несколько примеров. Для этого будем использовать демонстрационную базу «Конструирование форм».

В демонстрационной базе определены несколько ролей. Для каждой из ролей настроены те или иные права доступа на объекты конфигурации. Кроме того, как уже было сказано в начале главы, платформа предоставляет возможность настройки прав доступа на реквизиты объектов, на табличные части, на стандартные реквизиты, на состав команд формы, на командный интерфейс формы, на реквизиты формы и ее элементы.

Пример 1

Роли Полные права конфигурации на справочник Номенклатура даны все возможные права (рис. 2.151).

С помощью команд командной панели формы списка номенклатуры пользователь с ролью Полные права может делать с элементами и группами справочника все что угодно (рис. 2.152).

Роли Полные права доступно право Интерактивное добавление, и в пользовательском интерфейсе формы списка справочника есть команды создания и копирования. Право Интерактивная пометка на удаление дает возможность помечать объекты, предназначенные к удалению, и соответствующая кнопка командной панели доступна. Право Редактирование дает возможность изменять данные объекта информационной базы и сохранять эти изменения.

Если разработчику прикладного решения необходимо ограничить пользователя в его возможностях по интерактивному редактированию состава информационной базы, то следует прибегнуть к настройке ролей.

Ограничение по составу возможностей редактирования данных представлено для роли Менеджер по продажам (рис. 2.153).

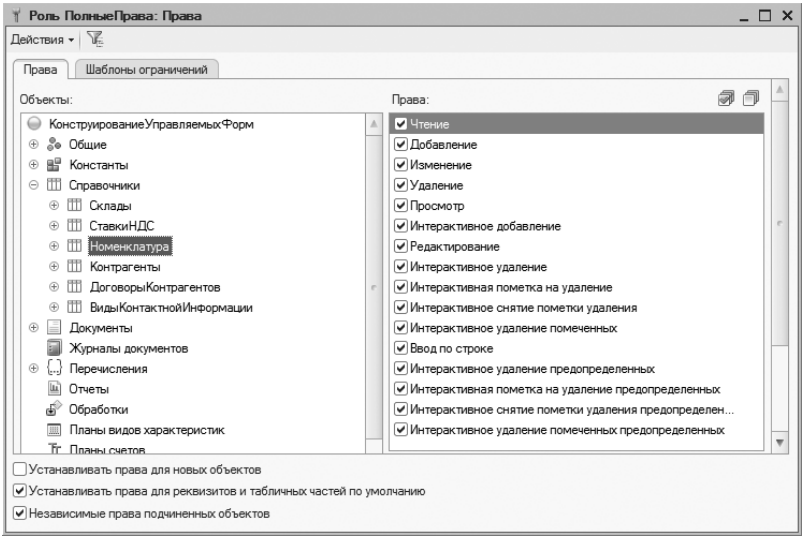


Рис. 2.151. Справочник «Номенклатура». Права «Полные права»

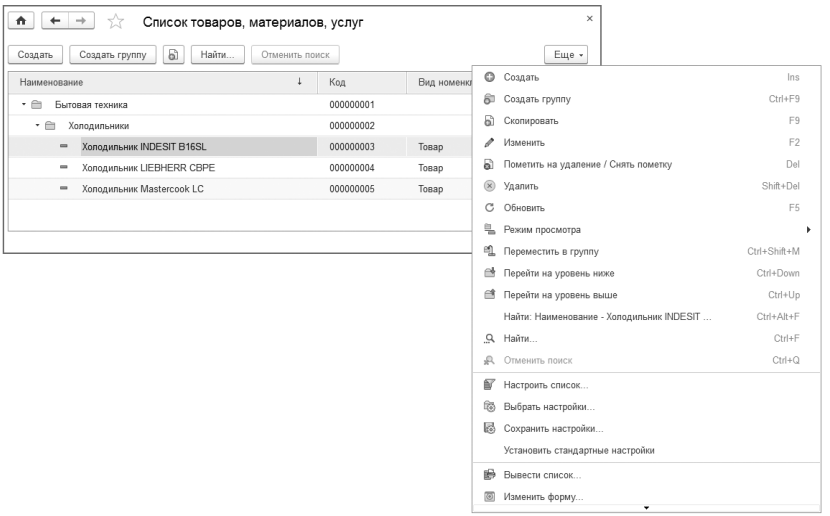


Рис. 2.152. Командная панель формы для пользователя с ролью «Полные права»

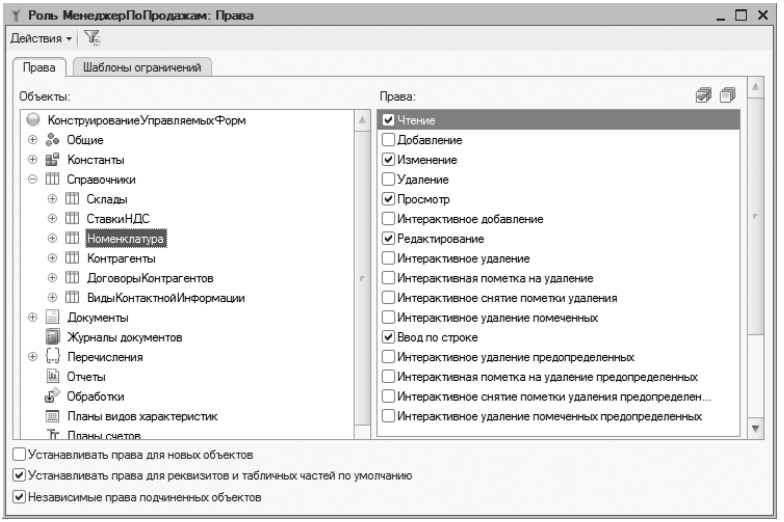


Рис. 2.153. Справочник «Номенклатура». Права «Менеджер по продажам»

Как видно из рисунка, пользователю с такими правами разрешено просматривать и редактировать данные базы данных. Результат влияния прав доступа пользователя на состав командных панелей формы можно увидеть, осуществив запуск прикладного решения под пользователем с правами Менеджер по продажам (рис. 2.154).

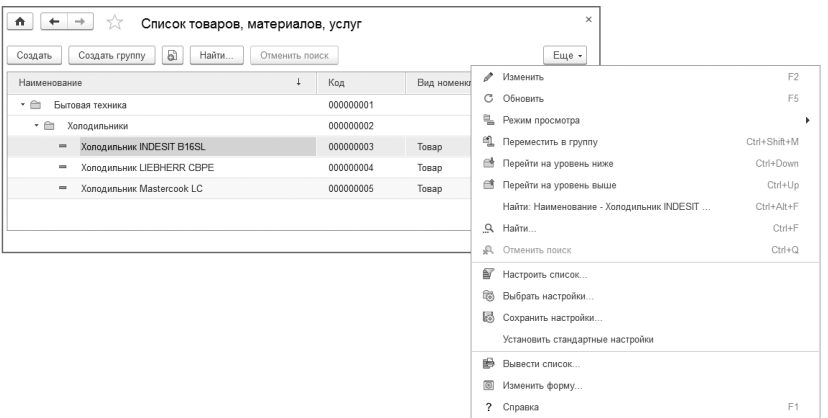


Рис. 2.154. Командная панель формы для пользователя с ролью «Менеджер по продажам»

Из командной панели формы исчезли команды, связанные с созданием, копированием, удалением новых объектов базы данных.

Пример 2

В демонстрационной базе роли Полные права установлены права доступа к регистру сведений Контактная информация. Поэтому в панели навигации окна клиентского приложения, в котором открывается форма элемента справочника Контрагенты, присутствует команда перехода к записям регистра Контактная информация, связанным с текущим контрагентом (рис. 2.155).

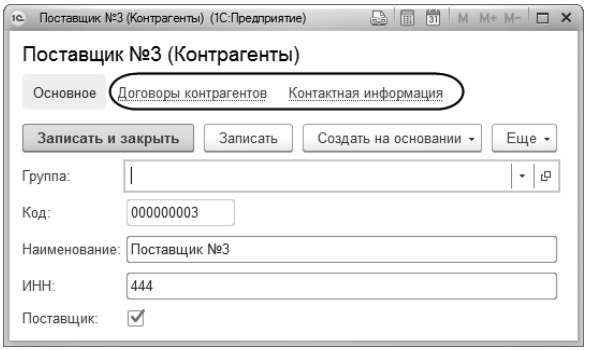


Рис. 2.155. Состав команд панели навигации для роли «Полные права»

Роли Менеджер по продажам на регистр сведений Контактная информация никакие права не назначены, и потому платформа ограничила состав команд панели навигации окна клиентского приложения (рис. 2.156).

Таким образом, разработчик прикладного решения имеет возможность изменять состав командных панелей формы, не прибегая к созданию «индивидуальных» форм, для всех определенных в конфигурации ролей. Для достижения нужного эффекта достаточно лишь правильно раздать роли и настроить права доступа.

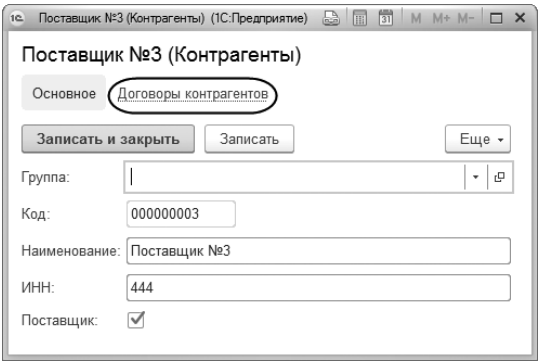


Рис. 2.156. Состав команд панели навигации для роли «Менеджер по продажам»

Пример 3

Настройки ролей и прав доступа влияют не только на состав командных панелей формы. Точно такому же воздействию подвергаются и все дочерние элементы формы, связанные с реквизитами объекта конфигурации или его табличными частями.

Такое влияние прав доступа можно увидеть на примере формы элемента справочника Номенклатура. Так, для роли Полные права установлены полные права доступа к табличной части справочника Поставщики, а для роли Менеджер по продажам такие права не установлены (рис. 2.157).

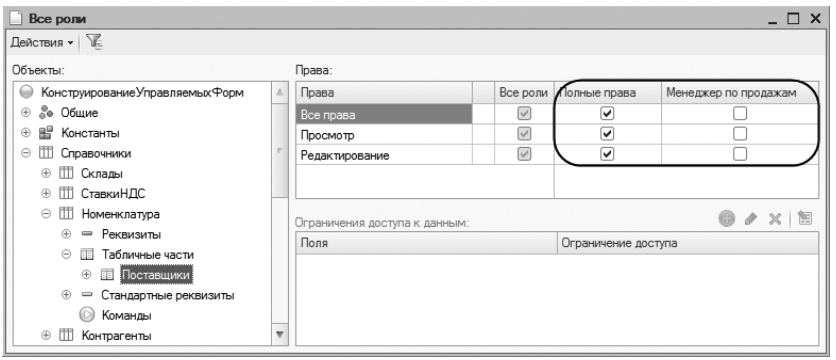


Рис. 2.157. Права доступа к табличной части

В пользовательском режиме работы тем пользователям, которым доступна роль Полные права, форма элемента справочника будет представлена в полном виде (рис. 2.158), а вот пользователям с ролью Менеджер по продажам придется только мечтать о возможности просматривать и редактировать табличную часть Поставщики (рис. 2.159).

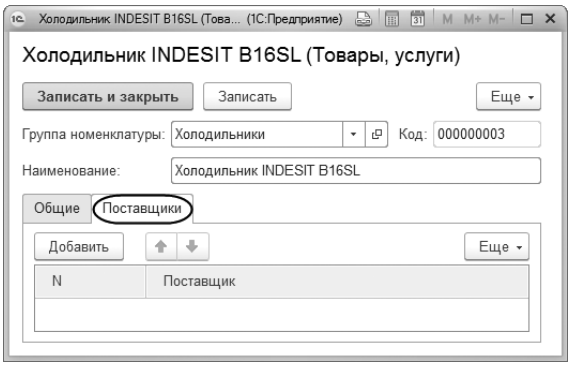


Рис. 2.158. Форма элемента. «Полные права»

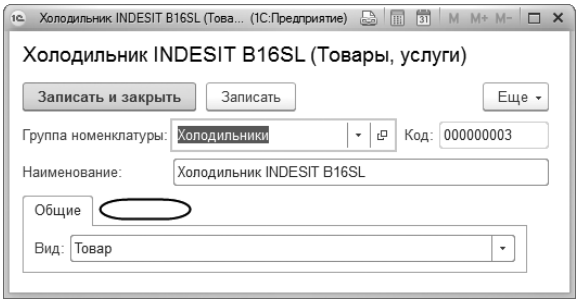


Рис. 2.159. Форма элемента. «Менеджер по продажам»

Если внимательно рассмотреть форму, представленную на рис. 2.159, то можно заметить, что вместе с табличной частью Поставщики исчезла и страница, на которой разработчик поместил эту табличную часть в редакторе формы. Произошло это потому, что при проектировании подчиненных элементов страницы формы на ней ничего, кроме табличной части, размещено не было (рис. 2.160). Таким образом,

определив состав страницы панели, платформа скрыла от пользователя и бесполезную для него страницу формы.

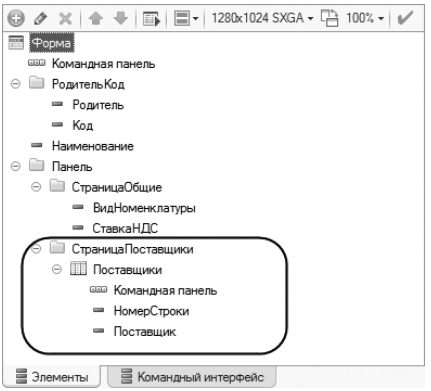


Рис. 2.160. Состав страницы формы

Скрытие платформой в пользовательском режиме работы других элементов формы (поля ввода, флажки и др.) происходит по точно такому же принципу, который был продемонстрирован выше.

Пример 4

Возможности изменения форм в зависимости от настройки прав доступа не ограничиваются только управлением видимостью элементов. В зависимости от предоставленных прав доступа платформа может просто запретить редактирование данных, оставив их при этом на экране пользователя.

Например, роли Менеджер по продажам запрещено редактировать реквизит Комментарий документа ПоступлениеТоваров (рис. 2.161).

В пользовательском режиме работы для менеджера по продажам элемент формы Комментарий будет присутствовать в форме, однако отредактировать его значение невозможно (рис. 2.162).

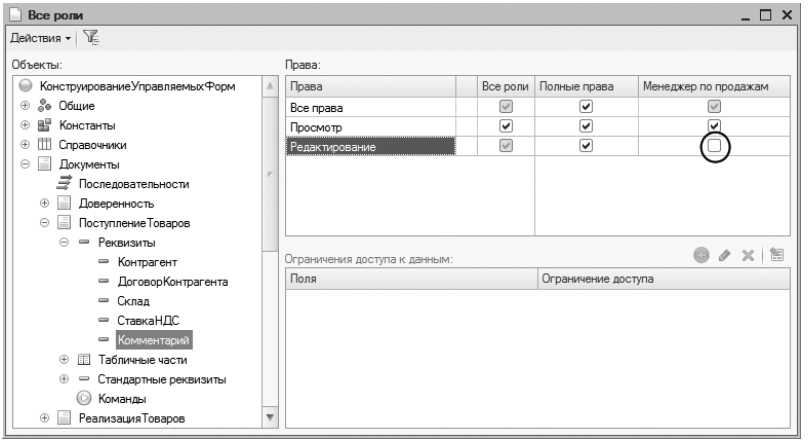


Рис. 2.161. Отключение права «Редактирование»

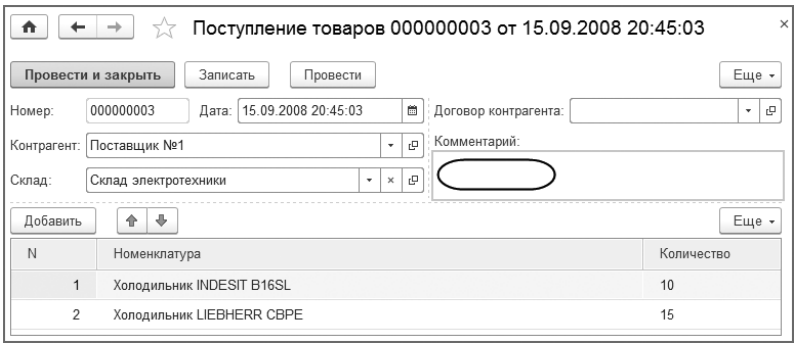


Рис. 2.162. Доступность реквизитов объекта

Влияние функциональных опций на элементы формы

Функциональные опции предназначены для оперативного управления функционалом прикладного решения в процессе его внедрения или работы без изменения конфигурации. Создание функциональных опций разработчиком осуществляется на этапе разработки конфигурации. В форме функциональные опции могут влиять на ее реквизиты, а значит, и на элементы формы (поля ввода, таблицы, страницы и прочие), связанные с этими реквизитами, и на команды формы, а значит, на кнопки формы, кнопки командных панелей.

В демонстрационной базе определена одна функциональная опция Учет НДС. Как ясно из названия опции, она отвечает за возможность ведения учета НДС в прикладном решении.

Реквизиты объектов конфигурации, табличные части и прочие элементы формы связываются с определенными функциональными опциями в соответствующем редакторе.

Пример 1

Рассмотрим влияние функциональных опций на форму на примере документа Реализация товаров.

Функциональная опция Учет НДС связана с реквизитами табличной части Товары: Ставка НДС, Сумма НДС. Для включения объектов конфигурации в состав функциональной опции нужно открыть ее состав и отметить флажком те объекты и их реквизиты, которые, по мнению разработчика (или по требованию заказчика), должны быть связаны с функциональной опцией (рис. 2.163).

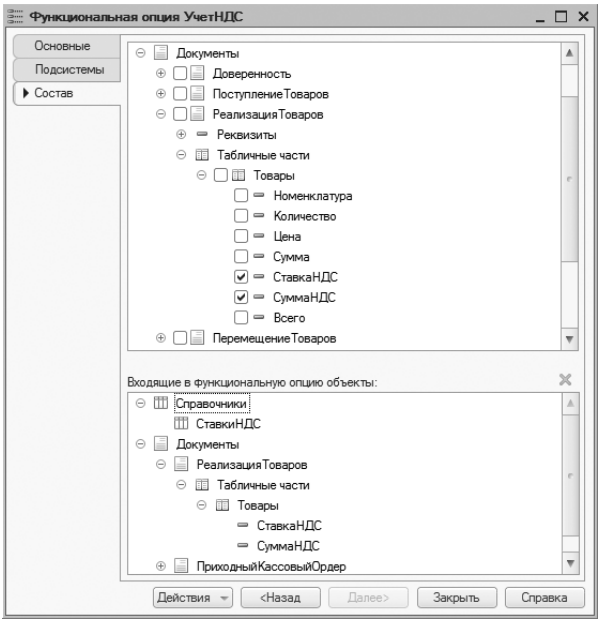


Рис. 2.163. Назначение функциональных опций реквизитам

Кроме того, в нашей демонстрационной базе функциональная опция Учет НДС связана со справочником Ставки НДС, что, впрочем, вполне логично для прикладного решения.

Таким образом, управление функциональной опцией Учет НДС будет оказывать влияние не только на формы приложения, но и на его интерфейс, о чем рассказывается в главе 1.7 на стр. 116.

Влияние функциональных опций на команды, выполняемые в форме, рассмотрим на примере глобальной команды Ставки НДС (для открытия списка соответствующего справочника), которая связана с кнопкой Открыть ставки НДС, помещенной в командной панели табличной части Товары формы документа Реализация товаров (рис. 2.164).

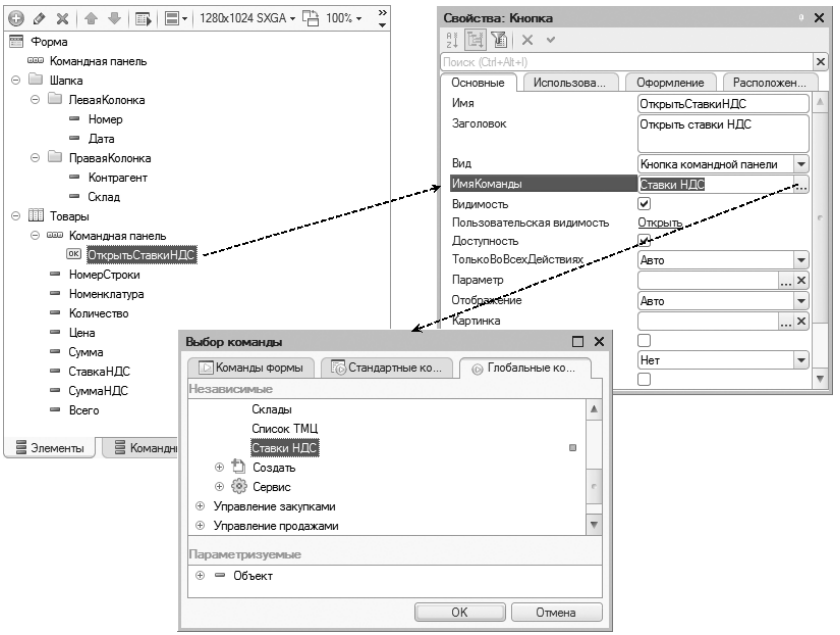


Рис. 2.164. Косвенная связь кнопки с функциональной опцией

Таким образом, задана косвенная связь кнопки командной панели таблицы формы с функциональной опцией. Несмотря на то что связь косвенная, функциональная опция окажет свое влияние на видимость кнопки.

По умолчанию в пользовательском режиме работы мы не увидим ни кнопки, ни колонок табличной части, связанных с функциональной опцией Учет НДС (рис. 2.165).

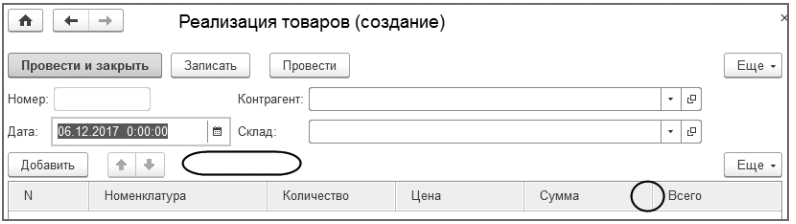


Рис. 2.165. Документ «Реализация товаров» по умолчанию

Отсутствие элементов формы, связанных с функциональной опцией Учет НДС, объясняется тем, что соответствующая функциональная опция в прикладном решении отключена (рис. 2.166).

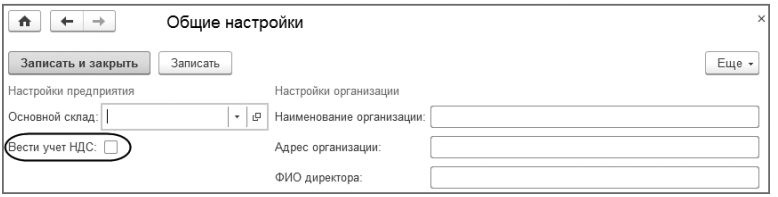


Рис. 2.166. Состояние функциональной опции «Учет НДС»

Установив флажок Вести учет НДС и записав изменения в информационную базу, мы тем самым включим функциональную опцию Учет НДС, и функционал, связанный с ней, будет доступен пользователям прикладного решения (рис. 2.167).

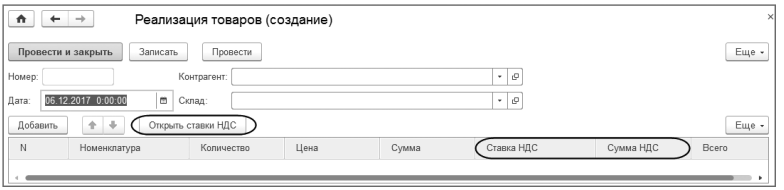


Рис. 2.167. Документ «Реализация товаров»

Пример 2

Связывать между собой кнопки формы (командных панелей) и функциональные опции можно не только косвенно. Как уже можно было заметить, связь осуществляется не с самой кнопкой, а с командой, которая будет выполняться при ее нажатии.

В качестве примера рассмотрим форму элемента справочника Номенклатура. В форме существует команда ЗаполнитьСтавкуНДС, связанная с функциональной опцией Учет НДС. В дереве элементов формы существует кнопка Заполнить ставку НДС (в виде гиперссылки), связанная с этой командой (рис. 2.168).

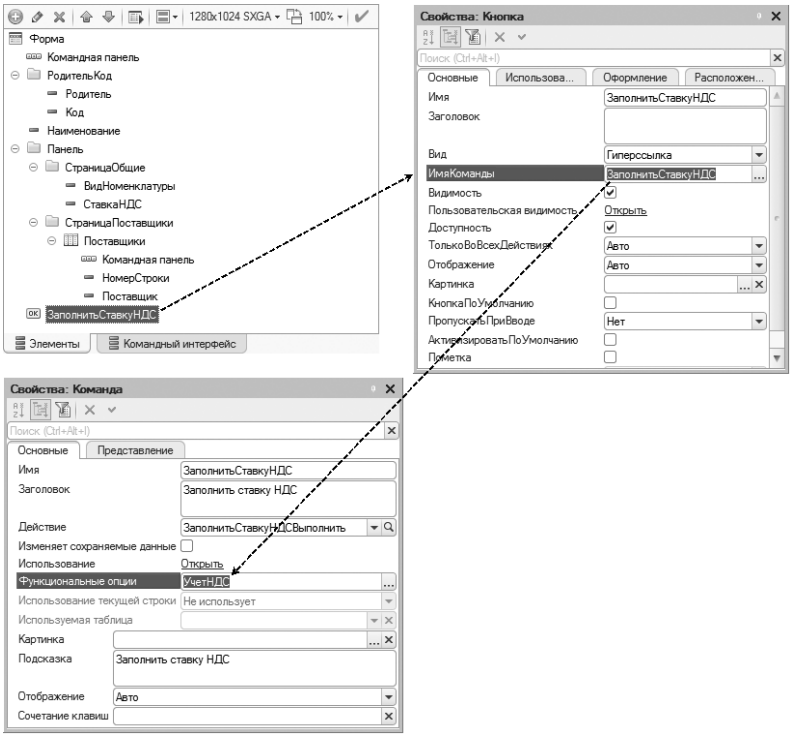


Рис. 2.168. Связь команды с функциональной опцией

Так как команда, выполняемая при нажатии гиперссылки Заполнить ставку НДС, связана с функциональной опцией, то, естественно,

гиперссылка будет доступна только при включенной функциональной опции Учет НДС (рис. 2.169).

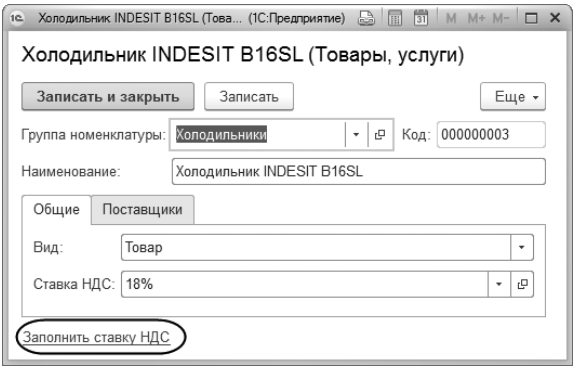


Рис. 2.169. Доступность элементов формы, связанных с функциональной опцией

При отключенной функциональной опции Учет НДС связанная с ней кнопка будет отсутствовать в форме номенклатуры (рис. 2.170).

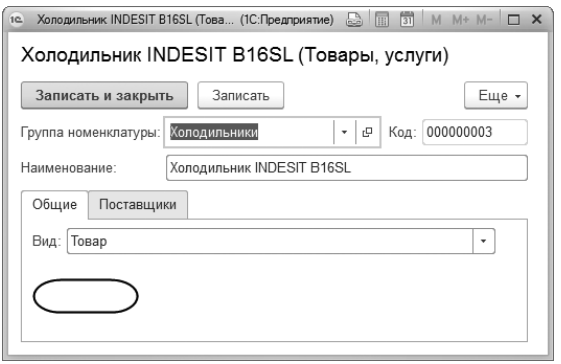


Рис. 2.170. Недоступность элементов формы, связанных с функциональной опцией

Кроме того, мы видим, что из формы «исчезло» и поле Ставка НДС, предназначенное для ввода и отображения реквизита СтавкаНДС справочника Номенклатура (см. рис. 2.169). Так произошло потому, что этот реквизит имеет тип ссылки на справочник СтавкиНДС, отключенный функциональной опцией.

Глава 2.8. Окно сообщений клиентского приложения

Окно сообщений формы предназначено для информирования пользователя о различных событиях. В процессе работы прикладного решения окно появляется в двух случаях:

- Сообщения сгенерированы программным способом. Вся работа по созданию таких сообщений целиком и полностью лежит на разработчике прикладного решения. Для вывода сообщений используются команды встроенного языка (см. главу 3.12 на стр. 553).
- Сообщения возникли вследствие попытки сохранить объект, не заполнив при этом обязательный для заполнения реквизит. Такие сообщения генерируются системой автоматически, на основании настроек, сделанных разработчиком прикладного решения в процессе редактирования свойств формы или ее дочерних элементов.

В этой главе рассматривается второй случай появления окна сообщений, так как именно такой вариант возникновения окна сообщений настраивается средствами визуального конструирования форм.

Для того чтобы сделать реквизит объекта обязательным для заполнения, необходимо изменить его свойство Проверка заполнения со значения Не проверять, устанавливаемого платформой по умолчанию, на значение Выдавать ошибку (рис. 2.171).

В пользовательском режиме работы поле, обязательное для заполнения, выделяется с помощью красной линии подчеркивания (рис. 2.172). Обязательным для заполнения можно сделать не только реквизит объекта, но и его табличную часть. В этом случае системой контролируется наличие хотя бы одной строки в проверяемой табличной части.

При попытке сохранения объекта системой генерируется сообщение об ошибке, которое показывается в окне сообщений внизу рабочей области окна. Одновременно с окном сообщений открывается окно с указателем на поле, в котором допущена ошибка (рис. 2.173, 2.174).

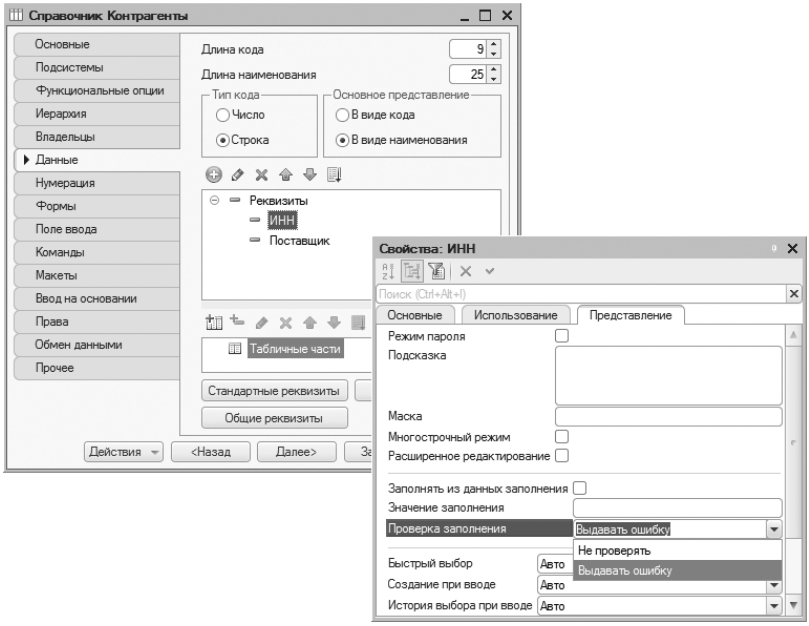


Рис. 2.171. Настройка проверки заполнения реквизита

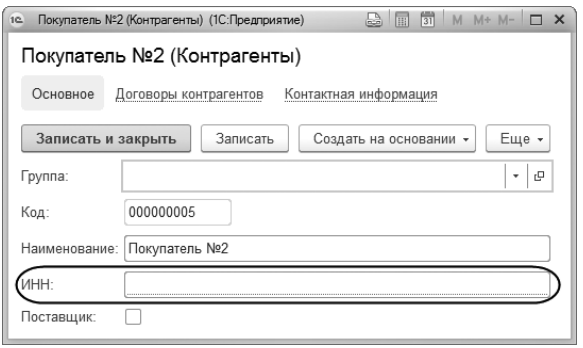


Рис. 2.172. Поле, обязательное для заполнения

В случае наличия нескольких ошибок переход по окнам с указателями на ошибочные поля осуществляется с помощью стрелок навигации.

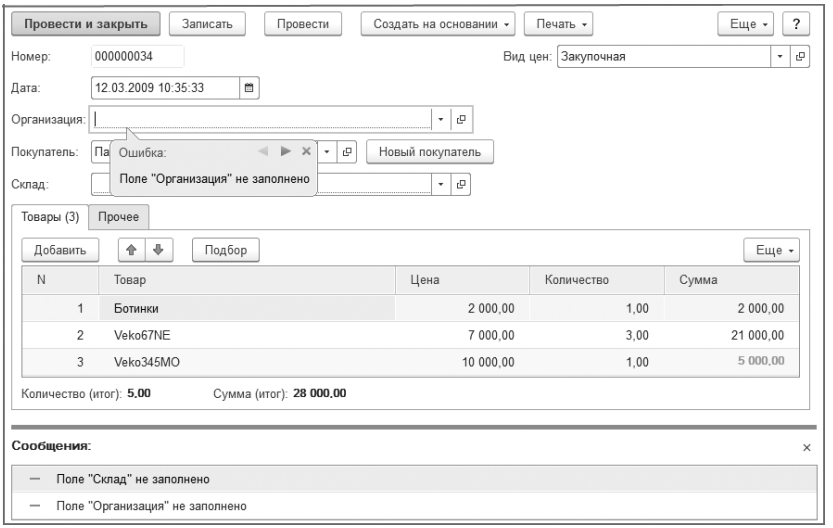


Рис. 2.173. Сообщения об ошибках

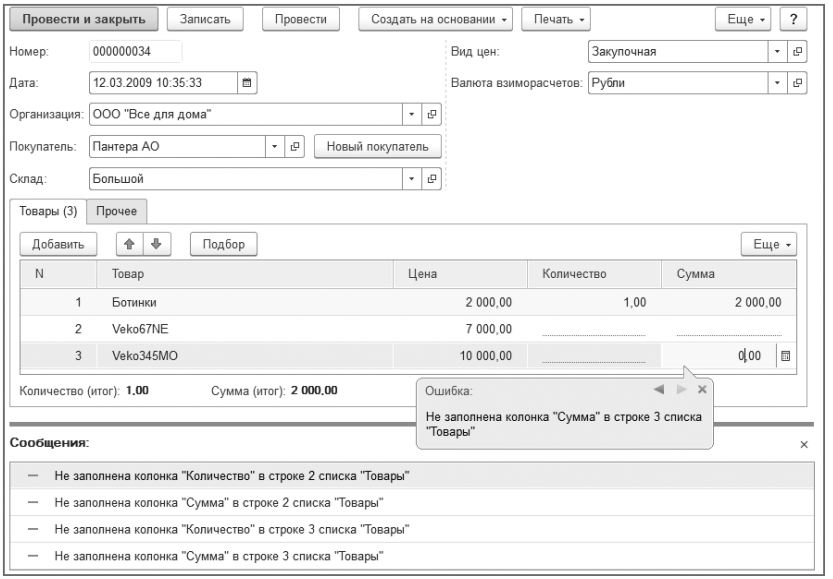


Рис. 2.174. Сообщения об ошибках заполнения табличной части

В случае попытки повторного сохранения или проведения объекта окно сообщений будет очищено и сообщения об ошибках будут сгенерированы снова.

Таким образом, система сама контролирует правильность заполнения обязательных данных и препятствует возникновению ошибок в базе данных, освобождая при этом разработчика прикладного решения от программирования реакции на ошибочные действия пользователя.

Глава 2.9. Примеры конструирования форм

Рассмотрение вопроса построения и модификации форм будет производиться на сквозном примере. В качестве эталона нами будет использован документ РасходныйКассовыйОрдер, который присутствует в демонстрационной базе «Конструирование форм».

Как и зачем объединять элементы формы в группы

По умолчанию форма документа, созданная конструктором форм (рис. 2.175), вряд ли может являться образцом изыска и удобства работы. Это и не входит в задачи конструктора форм. Его задача – сгенерировать форму, пригодную для ввода, редактирования, просмотра данных. С этой задачей конструктор справился блестяще.

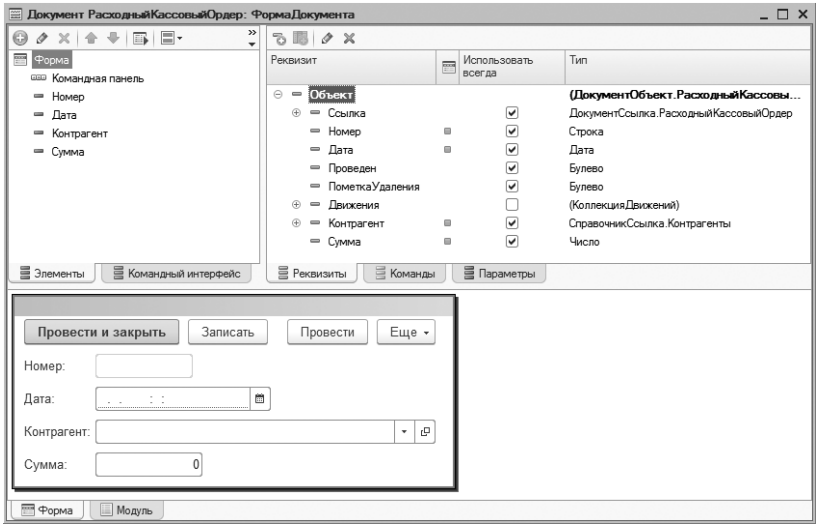


Рис. 2.175. Форма документа «Расходный кассовый ордер», созданная конструктором

Создадим еще одну форму документа ФормаДокумента1, назначим ее основной и все дальнейшие примеры будем выполнять на ней, чтобы к исходной форме документа можно было вернуться в случае необходимости.

Итак, чем же может помочь объединение элементов в группы? Сделать форму более читаемой, более удобной для просмотра и работы. Сгруппировать элементы формы в логические группы, объединив в них близкие, согласно логике работы приложения и формы, связанные данные. Это позволит рассматривать форму как некий каталог, в котором каждая группа (в нашем случае группа элементов) объединяет в себе близкие по характеристикам и назначению данные.

Выделим элемент формы Номер, чтобы добавленная группа располагалась прямо над ним. Добавим в форму элемент Группа – Обычная группа без отображения. Свойству Имя присвоим значение ДатаНомер.

Свойство Группировка стандартно установлено в значение Горизонтальная если возможно. Поскольку мы добавили обычную группу без отображения, то свойство ОтображатьЗаголовок уже отключено и свойство Отображение установлено в значение Нет. Что нам и нужно, так как в данном случае выводить название группы и выделять ее в форме не имеет ни малейшего смысла (рис. 2.176).

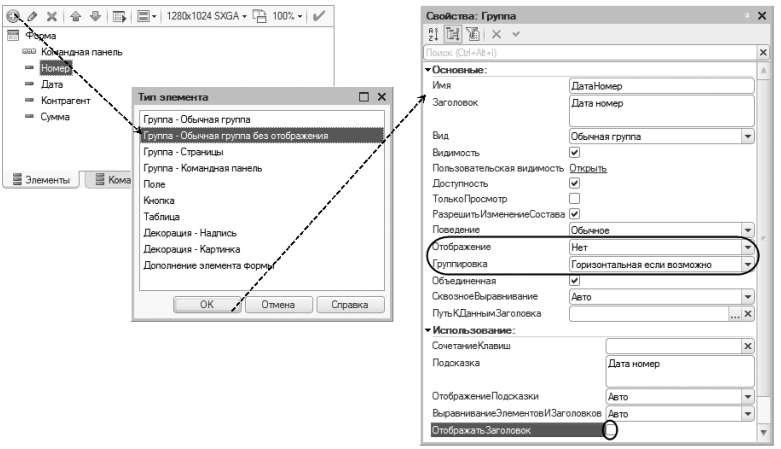


Рис. 2.176. Группировка элементов «Номер» и «Дата»

Подчиним группе элементы формы Номер и Дата и проверим результат. Итак, форма стала более приятной на вид (рис. 2.177).

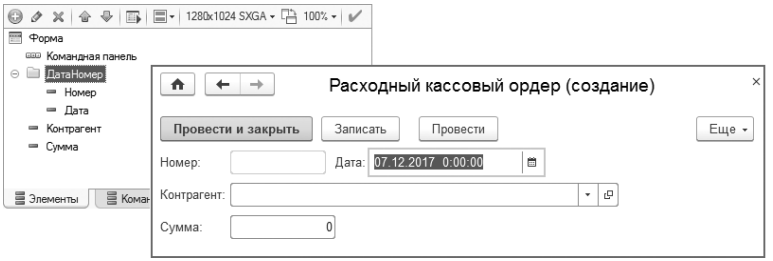


Рис. 2.177. Группировка элементов «Номер» и «Дата»

Все ли на этом? Нет.

Дело в том, что свойство РазрешитьИзменениеСостава у новой группы стандартно включено, поэтому пользователь в режиме 1С:Предприятие может изменять внешний вид этой формы по своему усмотрению, выполнив команду Изменить форму из подменю Еще.

Например, мой труд может выглядеть как на рис. 2.178 или как на рис. 2.179, а то и вообще как на рис. 2.180.

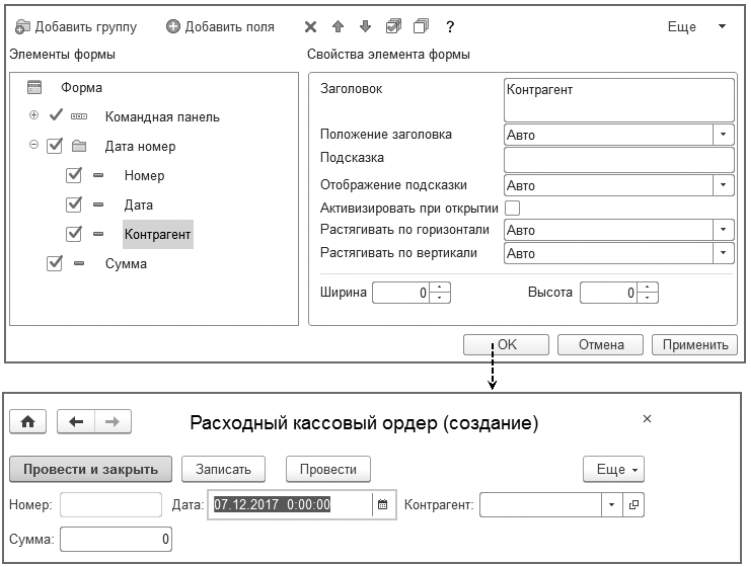


Рис. 2.178. Пользовательская настройка группы элементов 1

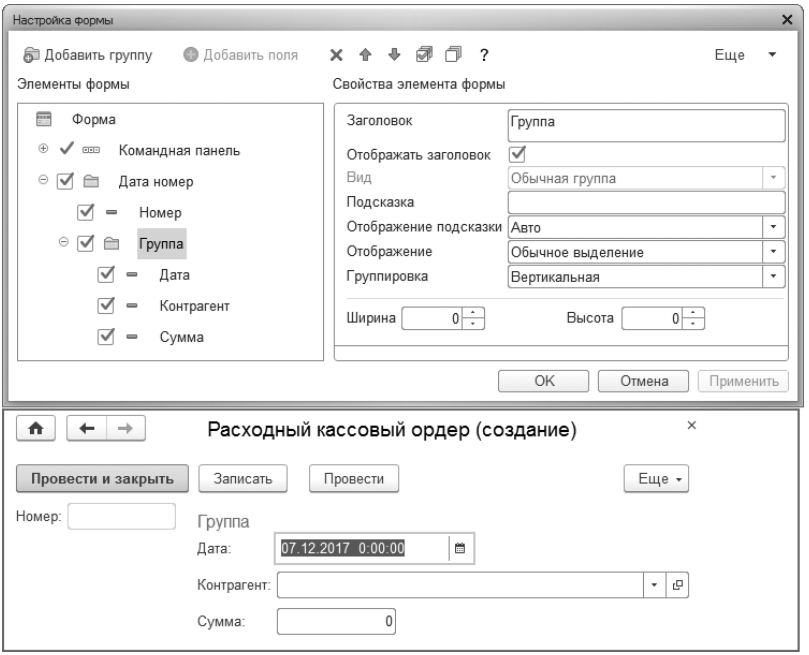


Рис. 2.179. Пользовательская настройка группы элементов 2

Может, некоторым разработчикам это и придется по нраву, но автор такого бы не пережил, потому сделаем так, чтобы состав группы ДатаНомер не изменялся пользователем в процессе настройки. Для этого свойство РазрешитьИзменениеСостава этой группы установим в Ложь (снимем флажок). Это не позволит пользователю добавлять в группу новые элементы или удалять существующие.

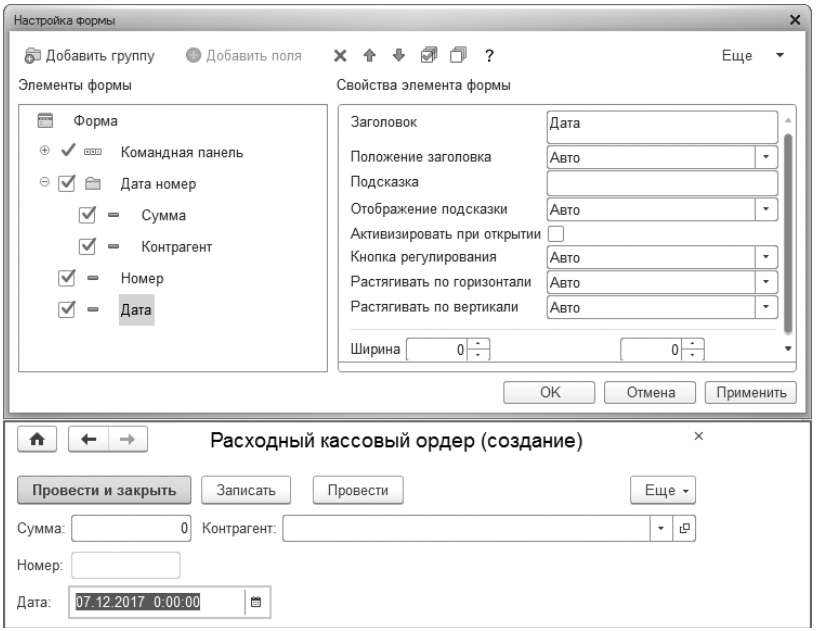


Рис. 2.180. Пользовательская настройка формы 3

Как изменить состав кнопок у элементов формы

Представим себя на месте пользователя нашей формы. Ему (пользователю) по неизвестным нам причинам необходимо очистить поле, в котором уже выбран контрагент.

Наш пользователь не читает документацию (то ли из-за отсутствия времени, то ли из-за лени, а может, и другие причины есть) и о горячих клавишах ничего не слышал. Кроме того, он не знает о наличии у элемента формы контекстного меню, в котором есть команда очистки. Поэтому пользователь хочет видеть кнопку очистки (в виде крестика) непосредственно в поле ввода контрагента.

Но дело в том, что автоматически кнопка очистки у поля ввода не отображается. И если пользователь настаивает на ее присутствии, необходимо лишь указать платформе на обязательное присутствие кнопки в элементе формы (рис. 2.181).

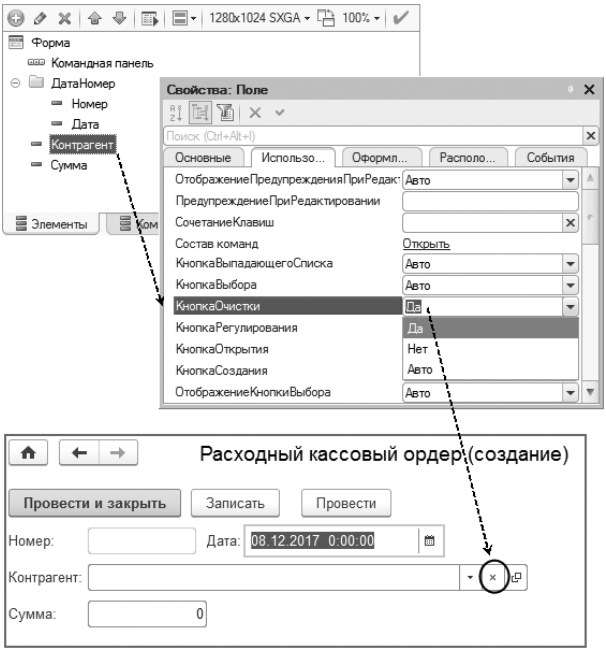


Рис. 2.181. Изменение состава кнопок элемента формы

Как добавить поле для ввода значений подчиненного справочника

Менеджеры вашего предприятия удачно приобрели несколько партий товаров, и руководство пришло к выводу, что оплаты поставщикам необходимо привязывать не только к контрагентам, но и договорам с ними (ибо товар приобретали каждый раз на разных условиях).

Договоры хранятся в справочнике, подчиненном контрагентам, то есть каждый контрагент имеет некоторое количество договоров. Именно из них (а не вообще всех договоров) пользователь и должен выбрать какой-то один.

Задача вам ясна, с теорией модификации формы вы уже знакомы по главе 2.3 на стр. 256, так что – вперед. Добавим в структуру данных документа реквизит ДоговорКонтрагента, разместим его в форме и ограничим пользователя в выборе договоров.

Итак, добавим реквизит ДоговорКонтрагента в документ РасходныйКассовыйОрдер и зададим тип этого реквизита – ссылка на справочник ДоговорыКонтрагента (рис. 2.182).

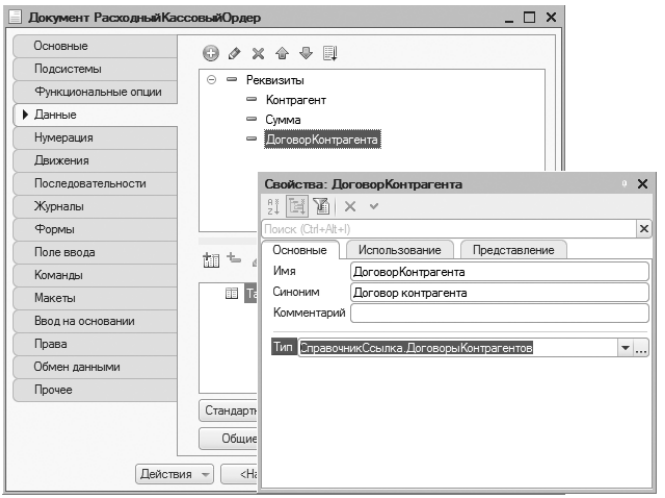


Рис. 2.182. Реквизиты документа

После этого реквизит станет доступен в форме документа в списке реквизитов основного реквизита формы. Перетащим его мышью в дерево элемента формы, при этом в форме появится соответствующее поле ввода, связанное с этим реквизитом (рис. 2.183).

Для получения положительного результата доработки осталось лишь добавить отбор договоров по контрагенту, и премия в кармане.

Чтобы список договоров отбирался с учетом выбранного ранее контрагента, необходимо задать этот отбор в свойстве СвязиПараметровВыбора реквизита ДоговорКонтрагента.

Однако воспользоваться этим свойством элемента формы, связанного с реквизитом, значит допустить методологическую ошибку. Если вы так и поступили, то вам стоит перечитать теоретические главы этой книги (главы 2.3 стр. 256, 2.4 стр. 327). Дело в том, что часть свойств элементов формы сейчас есть и у реквизитов объектов, а значит, более правильно настраивать это там.

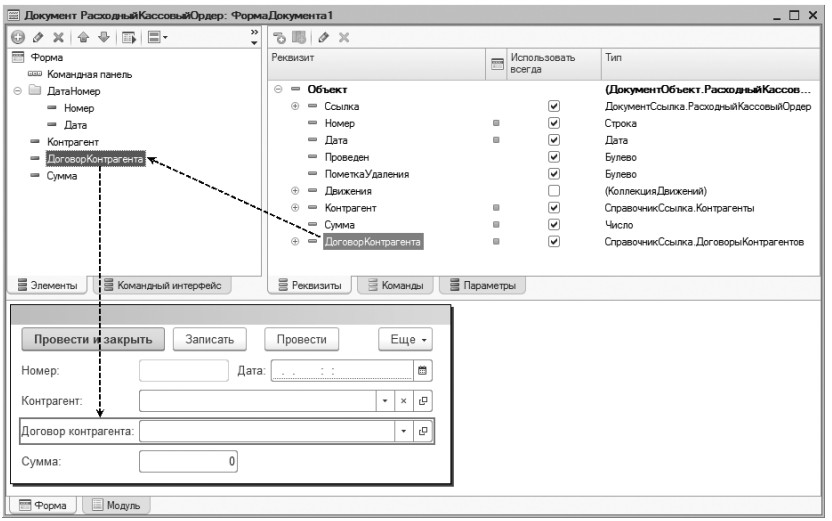


Рис. 2.183. Отображение в форме нового реквизита объекта

Откроем палитру свойств реквизита ДоговорКонтрагента и нажмем кнопку выбора у свойства СвязиПараметровВыбора. Откроется окно настройки параметров для выбора значений реквизита. Слева находится список всех доступных реквизитов документа, значения которых могут ограничивать список выбора для реквизита ДоговорКонтрагента. Перенесем в правый список параметров выбора реквизит Контрагент (рис. 2.184).

В поле Имя параметра выбора платформа автоматически добавила отбор по полю Владелец – `Отбор.Владелец`. Так произошло потому, что реквизит ДоговорКонтрагента имеет тип ссылки на справочник ДоговорыКонтрагента. А этот справочник подчинен справочнику Контрагенты, на который, в свою очередь, ссылается реквизит Контрагент, значения которого будут накладывать отбор на список договоров контрагента.

Как же все это работает? Да очень просто. Список договоров строится с помощью системы компоновки данных, на отбор которой и влияет свойство Связи параметров выбора (рис. 2.185).

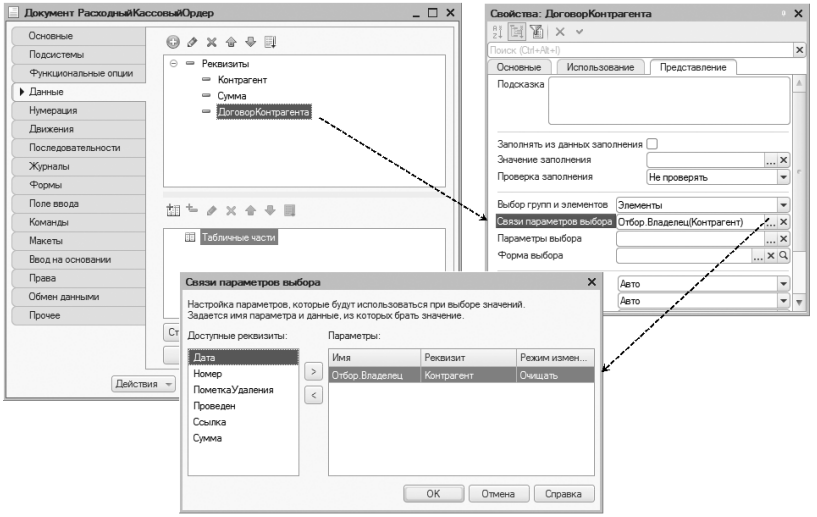


Рис. 2.184. Настройка отбора договоров по контрагенту

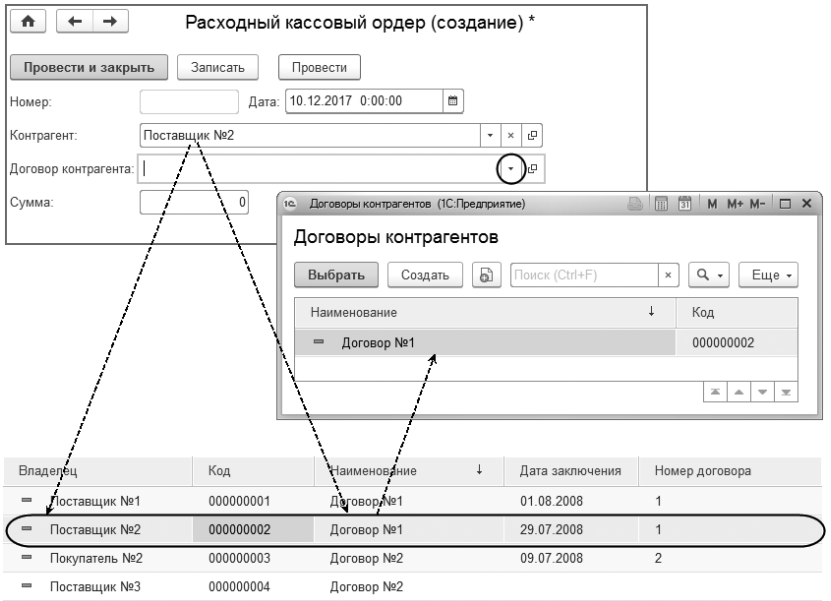


Рис. 2.185. Отбор договоров по контрагенту

Вполне очевидно, что во всех формах нашего документа нам будет необходим отбор договоров по текущему контрагенту. Вот поэтому мы и настроили его на уровне не одной формы, а на уровне реквизита объекта.

Как добавить в форму табличную часть

Поработав некоторое время с нашим документом, бухгалтерия вносит предложение: дать возможность закрывать одним документом несколько сделок, по нескольким договорам.

Способов решения поставленной задачи существует немало. Мы рассмотрим классический вариант с использованием табличной части документа.

В документ РасходныйКассовыйОрдер добавим табличную часть Оплаты. В табличной части должны быть реквизиты Договор (тип СправочникСсылка.ДоговорыКонтрагентов), ПриходныйДокумент (тип ДокументСсылка.ПоступлениеТоваров), Сумма.

ВНИМАНИЕ!

Если вы для реквизита Договор не заполнили свойство Связи параметров выбора, то сейчас самое время сделать это. Для чего это делается, рассказано в предыдущем разделе главы.

Самый простой способ добавления табличной части в форму – это перетаскивание из раздела редактора формы Реквизиты. Зацепим мышью табличную часть Оплаты и бросим в раздел Элементы.

Если вы не промахнулись, то платформа предложит вам автоматически создать колонки будущей таблицы. Можете соглашаться с этим. Исправить созданное можно всегда.

Вот, собственно, и все с точки зрения конструирования формы. Форма несколько непривычна (много реквизитов, таблица, а что к чему – непонятно), и неопытный пользователь наверняка в ней запутается (рис. 2.186). Как немного поправить ситуацию, расскажем далее.

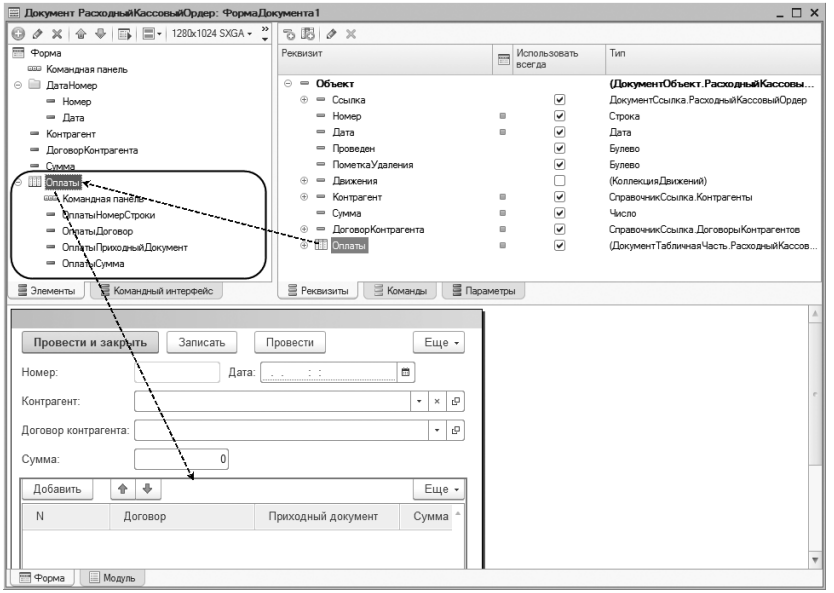


Рис. 2.186. Таблица, отображающая табличную часть документа

Как добавить в форму группу страниц

С помощью группы страниц можно улучшить дизайн и удобство работы с формой, сделать форму более компактной и читаемой. Неудобную, «сложную» форму документа, получившуюся после добавления в нее табличной части (см. рис. 2.186), мы «облегчим» за счет разбиения информации по страницам. На отдельную страницу поместим таблицу для редактирования табличной части документа.

Чтобы добавить в форму группу страниц, нужно добавить родительскую группу (элемент формы Группа, свойство Вид которого имеет значение Страницы) и вложить в нее дочерние группы вида Страница. А затем уже эти дочерние группы наполнить нужными элементами формы.

Добавим в дерево элементов формы новую группу, определив свойство Вид в значение Страницы (рис. 2.187).

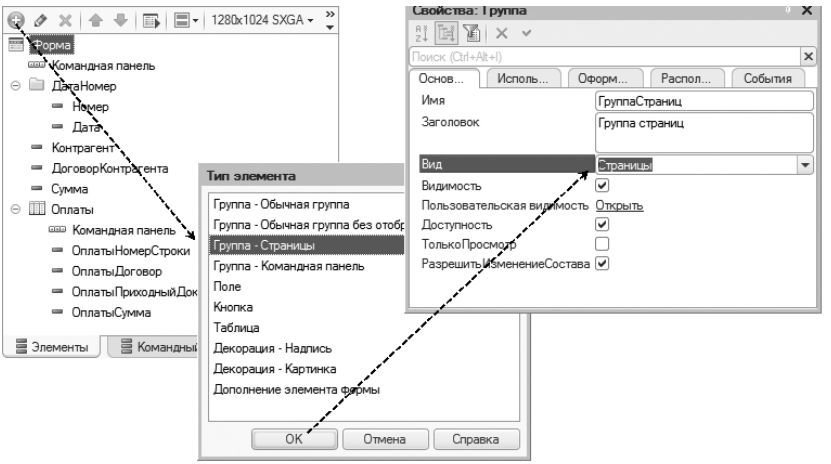


Рис. 2.187. Элемент «Группа» вида «Страницы»

Назовем ее ГруппаСтраниц, поместим ее над таблицей формы Оплаты и добавим в нее несколько подчиненных групп-страниц, в которых разместим элементы формы.

Добавим в подчинение группе ГруппаСтраниц группы ПлатежПоСделке и ПлатежиПоСделкам, свойство Вид которых будет иметь значение Страница (рис. 2.188).

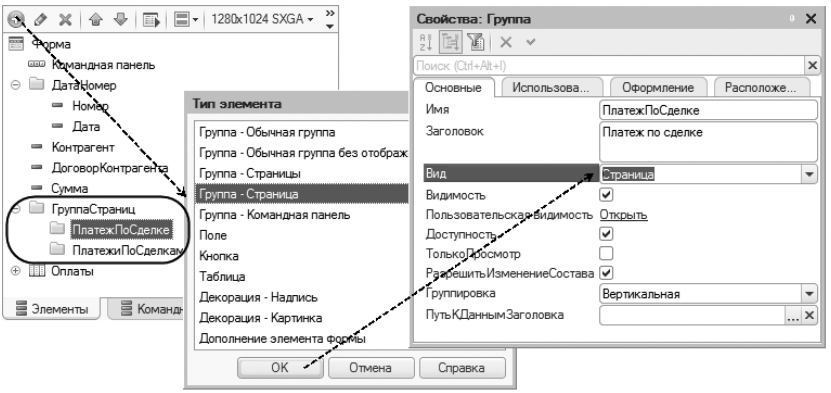


Рис. 2.188. Страницы формы

Для того чтобы страницы стали видны в форме, перегруппируем элементы формы так, чтобы элементы ДоговорКонтрагента и Сумма размещались на странице ПлатежПоСделке, а таблица Оплаты – на странице ПлатежиПоСделкам (рис. 2.189).

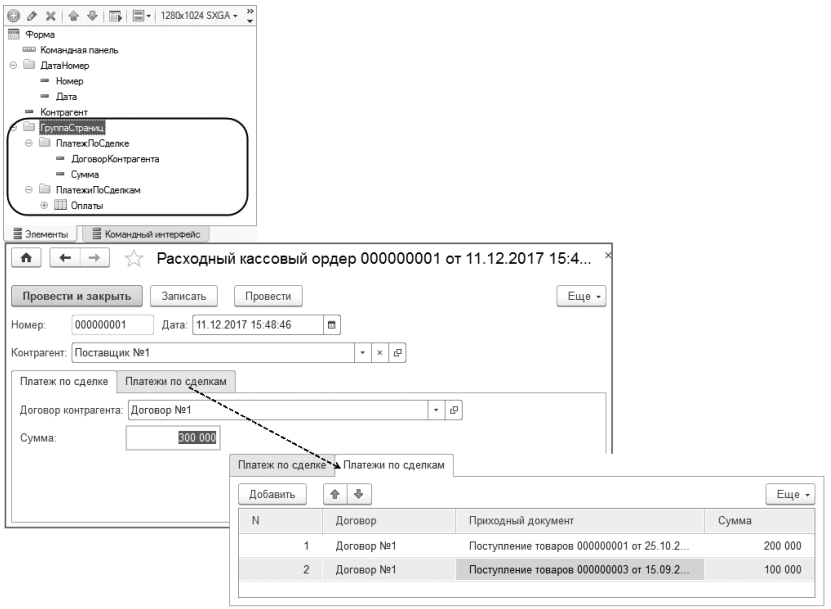


Рис. 2.189. Распределение элементов по страницам

Теперь информация в форме разбита на отдельные смысловые группы, за счет чего форма лучше смотрится и легче воспринимается.

Пожалуй, можно еще улучшить дизайн страницы, на которой расположена табличная часть Оплаты, если в подвале таблицы показывать итог по колонке Сумма, а в заголовке страницы выводить количество строк в таблице.

Для этого выделим в окне элементов формы таблицу Оплаты и установим в палитре ее свойств флажок Подвал. Затем выделим поле таблицы ОплатыСумма и в свойстве поля ПутьКДаннымПодвала выберем из списка реквизитов Объект.Оплаты.ИтогСумма (рис. 2.190).

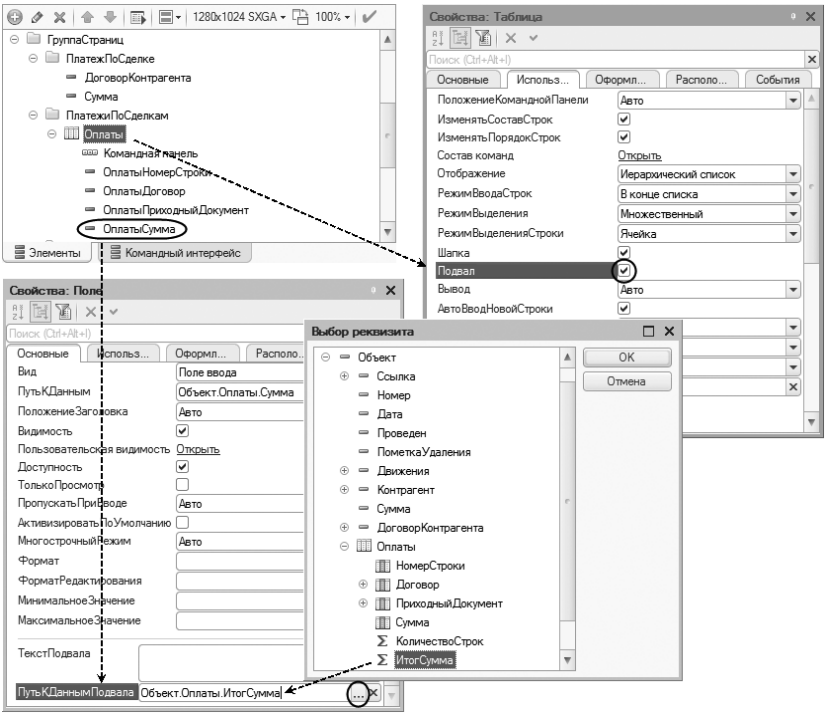


Рис. 2.190. Установка свойств таблицы «Оплаты»

Затем выделим в группе страниц формы страницу ПлатежиПоСделкам и в свойстве ПутьКДаннымЗаголовка выберем из списка реквизитов Объект.Оплаты.КоличествоСтрок (рис. 2.191).

В результате форма документа стала иметь еще более понятный и привлекательный вид (рис. 2.192).

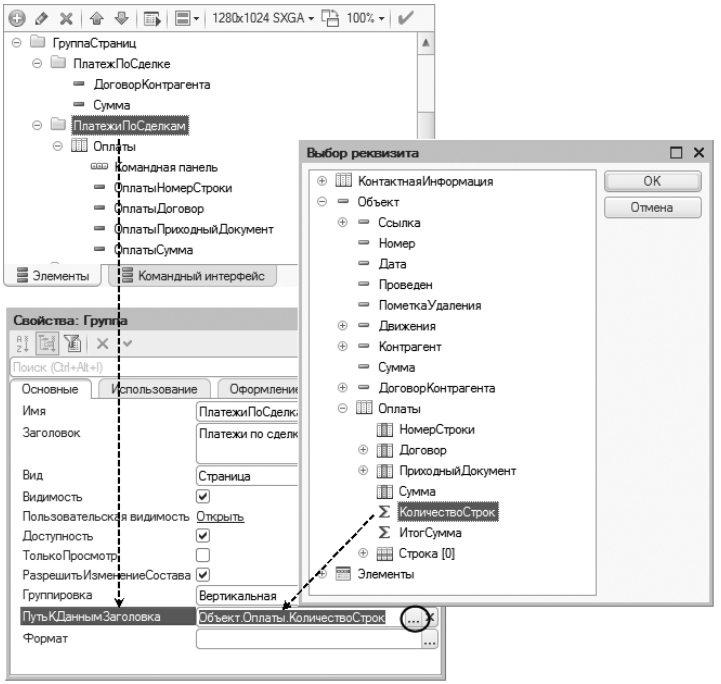


Рис. 2.191. Установка свойств страницы «Платежи по сделкам»

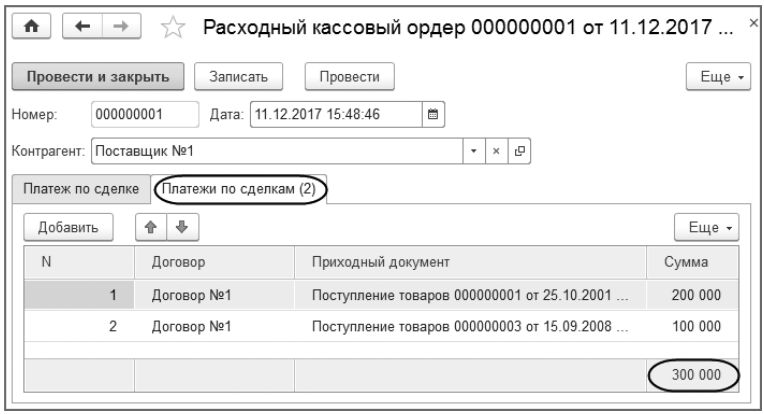


Рис. 2.192. Форма документа «Расходный кассовый ордер»

Как добавить в форму таблицу, отображающую связанные данные

Согласно регламенту работы компании, деятельность которой мы с вами взялись автоматизировать, после оплаты поставленных товаров (оплата в программе, а значит, и в банке) принято уведомлять об этом поставщиков. Таким образом, перед нами стоит задача добавить в форму контактные данные поставщика.

Раз перечень контактных данных жестко не определен (например, только рабочий телефон менеджера), то наиболее оптимальным для вывода заранее неизвестного количества данных является таблица.

Как нам уже известно, для того чтобы что-то отображалось в форме, это что-то должно быть описано в виде реквизита формы.

Добавим в форму новый реквизит КонтактнаяИнформация. Тип значения реквизита ДинамическийСписок (рис. 2.193).

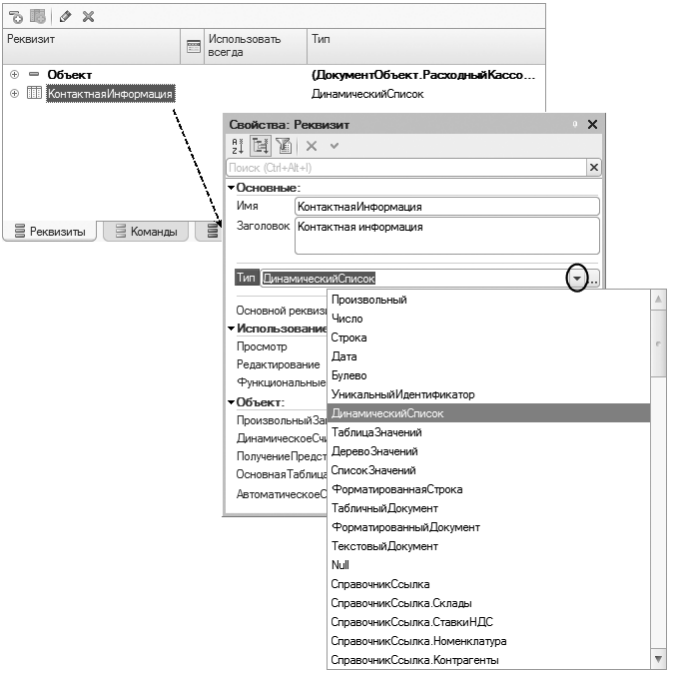


Рис. 2.193. Тип реквизита «ДинамическийСписок»

Динамический список может получать данные из таблиц базы данных. Это может быть, например, список справочника, список документов, список значений регистра сведений.

Для того чтобы динамический список работал с такими данными, необходимо указать их в свойстве ОсновнаяТаблица (рис. 2.194).

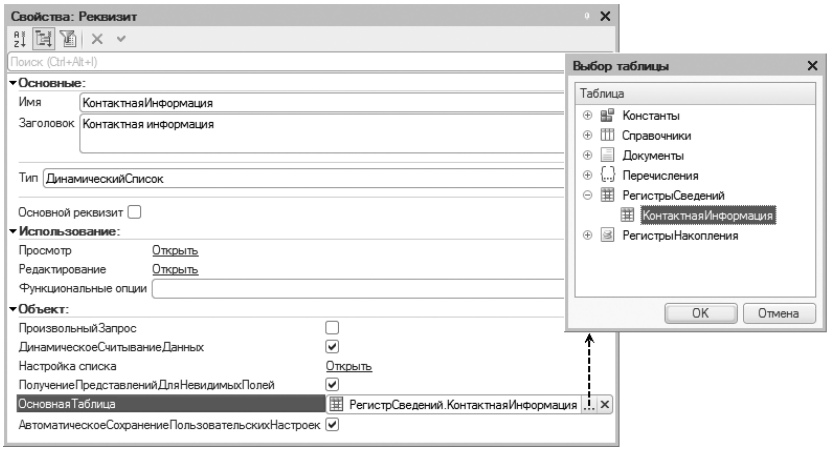


Рис. 2.194. Свойство «ОсновнаяТаблица»

С помощью свойства НастройкаСписка можно произвести необходимые настройки отбора, группировки, сортировки отображаемых данных.

Динамический список может получать данные из результата произвольного запроса, который необходимо определить в свойстве НастройкаСписка. Для того чтобы использовать произвольный запрос, необходимо установить флажок ПроизвольныйЗапрос (рис. 2.195).

Свойство Динамическое считывание данных отвечает за считывание данных из базы данных порциями, что позволяет экономить ресурсы системы.

Для решения поставленной перед нами задачи мы воспользуемся первым способом (задание основной таблицы, см. рис. 2.194) и выберем таблицу регистра сведений КонтактнаяИнформация.

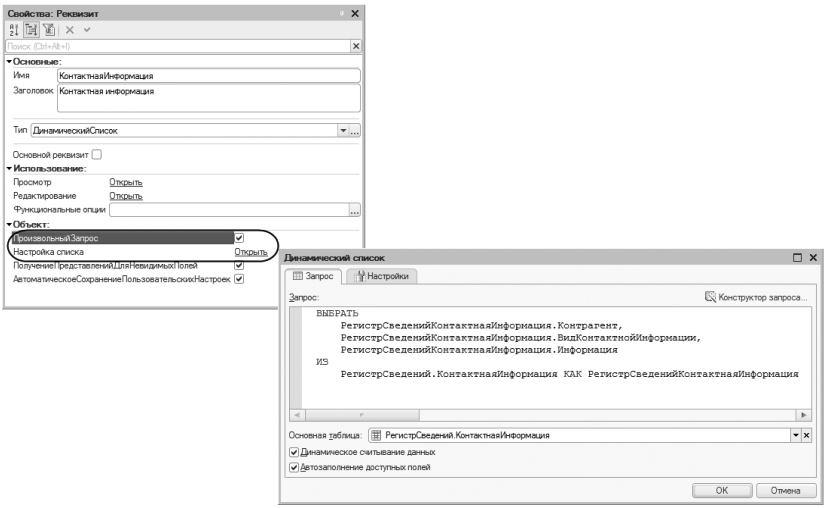


Рис. 2.195. Свойство «ПроизвольныйЗапрос»

Добавим в группу страниц формы еще одну страницу КонтактнаяИнформация и разместим на ней таблицу с контактной информацией контрагентов (рис. 2.196).

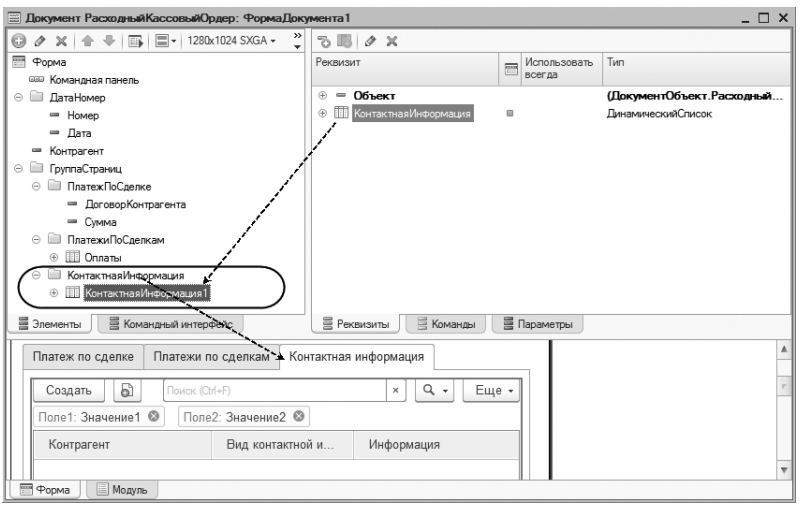


Рис. 2.196. Отображение контактной информации

В некоторых случаях использовать дополнительные страницы, таблицы и динамические списки вовсе не обязательно. Платформа позволяет выводить связанную с неким реквизитом информацию в дополнительную форму этого же окна клиентского приложения (в котором открыта основная форма объекта) и осуществлять переключение между этими формами с помощью панели навигации окна клиентского приложения.

Если открыть в редакторе формы Глобальные команды, то можно увидеть, что нам доступна параметризуемая команда открытия контактной информации, связанной с реквизитом Контрагент. Такую команду достаточно перетащить в панель навигации формы на закладке Командный интерфейс. Всю остальную работу за нас сделает система при построении и отображении формы на экране (рис. 2.197).

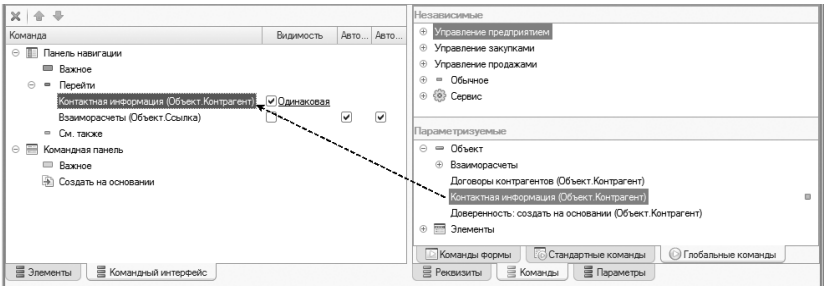


Рис. 2.197. Отображение контактной информации

Чтобы перейти к контактной информации контрагента, выбранного в документе, достаточно нажать на ссылку Контактная информация в панели навигации окна клиентского приложения (рис. 2.198).

Отбор, который был при этом установлен системой, не потребовал от разработчика ни строчки кода. В то время как в динамическом списке, отображающем контактные данные контрагентов, нужно программно устанавливать отбор по контрагенту, выбранному в документе.

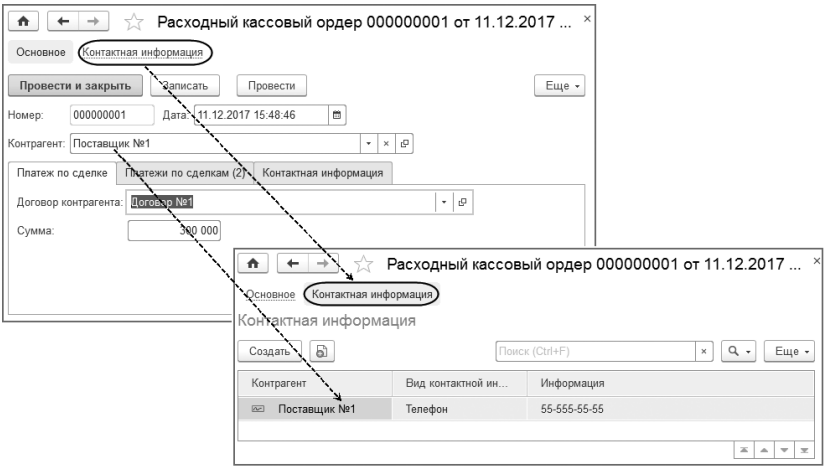


Рис. 2.198. Контактная информация текущего контрагента

Мы специально не стали акцентировать на этом внимание, так как речь о программном отборе в динамическом списке пойдет в третьей части книги в главе 3.15 «Оформление списков», раздел «Динамические списки» на стр. 584. Но вообще-то, чтобы в таблице контактной информации показывались только контакты текущего контрагента, нужно создать обработчик события ПриИзменении для поля документа Контрагент и заполнить его следующим образом (листинг 2.1).

Листинг 2.1. Обработчик события «ПриИзменении» поля «Контрагент»

```
&НаКлиенте
Процедура КонтрагентПриИзменении(Элемент)

    Отбор = КонтактнаяИнформация.Отбор.Элементы.Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
    Отбор.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("Контрагент");
    Отбор.ВидСравнения = ВидСравненияКомпоновкиДанных.Равно;
    Отбор.ПравоеЗначение = Объект.Контрагент;
    Отбор.Использование = Истина;
    ОбновитьОтображениеДанных();

КонецПроцедуры
```

И затем нужно продублировать содержимое обработчика в обработчик события формы ПриЧтенииНаСервере.

Если рассматривать наш пример с контактной информацией, то второй способ (с помощью команды панели навигации) является наиболее предпочтительным и рекомендуемым.

Как создать и заполнить объект с учетом установленного отбора списка

Достаточно часто у пользователя возникает необходимость проанализировать некие данные (например, существующие документы определенного контрагента) и, обнаружив расхождения, создать еще один документ. При этом часто копирование документа или ввод на основании не дают положительного эффекта – необходимо изменять достаточно большое количество реквизитов, и что-то можно просто оставить по ошибке, хотя нужно бы поменять.

Выходом из сложившейся ситуации может быть использование отборов в списке и установка флажка Заполнять из данных заполнения у нужных реквизитов объекта.

Итак, наш пользователь, получив от контрагента список оплат, просматривает тот же список в своей информационной базе (рис. 2.199).

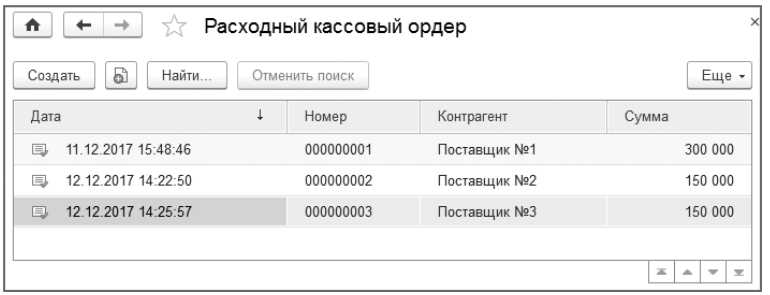


Рис. 2.199. Список документов «Расходный кассовый ордер»

Для лучшего восприятия данных пользователем устанавливается отбор по нужному контрагенту (рис. 2.200).

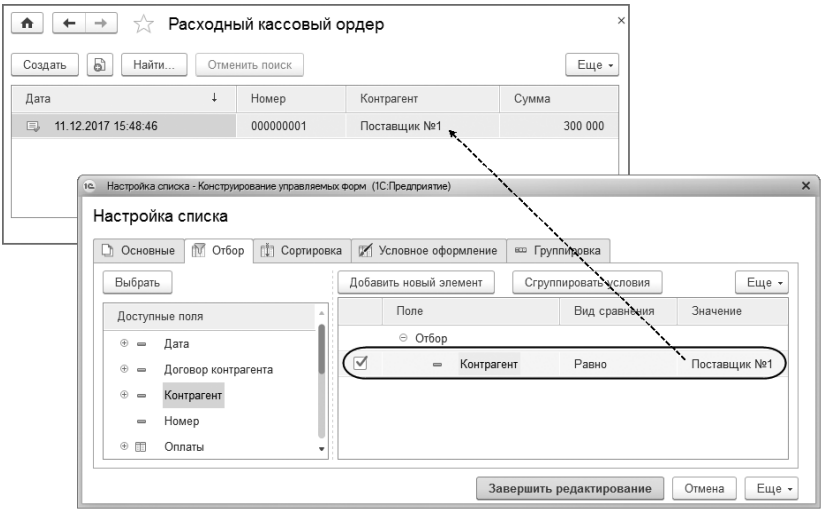


Рис. 2.200. Установленный пользователем отбор

Мы уже говорили о том, что сложная структура нашего документа не позволяет нам эффективно использовать методы копирования или ввода на основании (если это предусмотрено) документа. Пользователю необходимо лишь автоматизировать заполнение нескольких реквизитов.

Этого легко добиться, если установить у нужного реквизита документа свойство Заполнять из данных заполнения. Установим этот флажок у реквизита документа Контрагент (рис. 2.201).

После такой настройки новый объект конфигурации (наш документ) будет при создании заполнять свои реквизиты (у которых установлен флажок Заполнять из данных заполнения) исходя из установленных пользователем отборов списка (рис. 2.202).

ВНИМАНИЕ!

Заполнение будет происходить только при интерактивном создании документа (кнопка Создать). В списке должен быть установлен именно отбор. Если, например, пользователь применял поиск по полю Контрагент, то отбор в списке не устанавливается (рис. 2.203) и реквизит заполнен не будет.

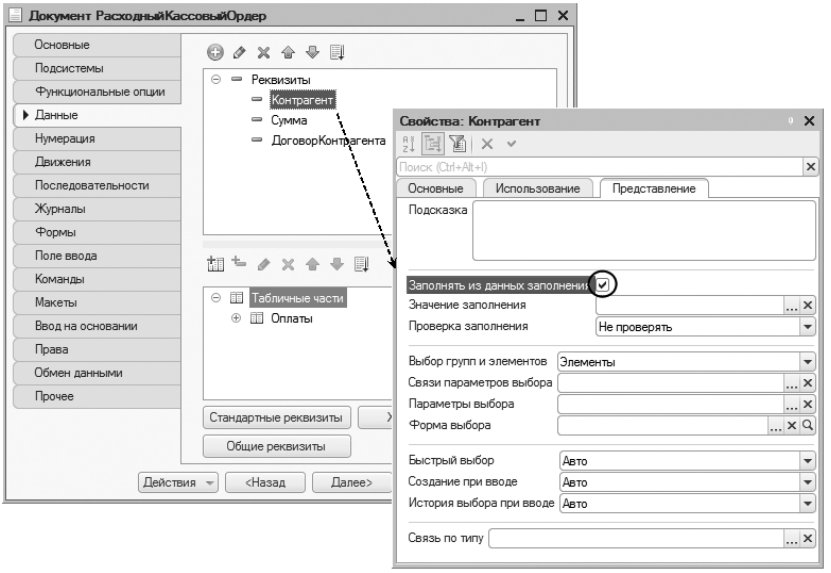


Рис. 2.201. Установка свойства «Заполнять из данных заполнения»

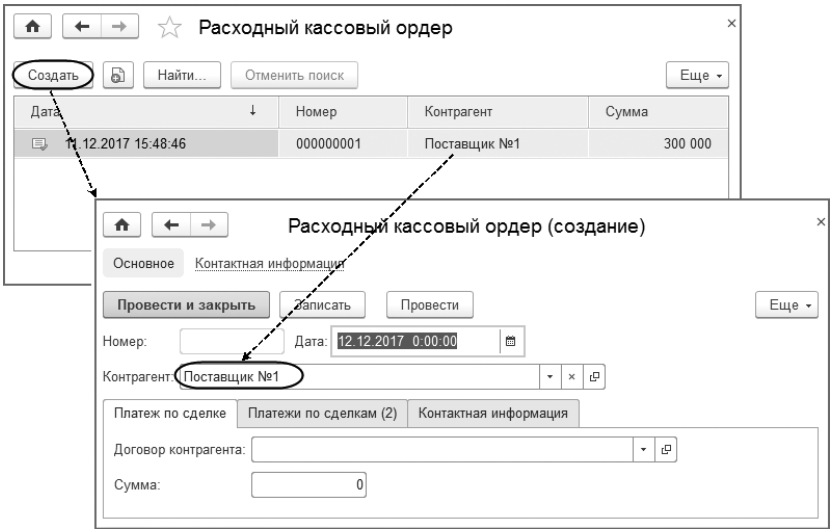


Рис. 2.202. Заполнение документа с учетом установленного отбора

Рис. 2.203. Результат поиска в списке

Как отобразить в списке реквизиты реквизитов

Предположим, нам нужно вывести в форму документа некоторые дополнительные данные о контрагенте, которому осуществлялся платеж. Например, мы хотим видеть в документе информацию, которая содержится в справочнике контрагентов, – об ИНН контрагента (строковой реквизит ИНН), а также признак того, является ли контрагент поставщиком (булевый реквизит Поставщик).

При этом требуется сделать просмотр этих данных опциональным, чтобы блок с информацией можно было свернуть или развернуть

по желанию пользователя. Из соображений компактности нужно, чтобы группа с дополнительной информацией была изначально свернута.

Добавить в форму «реквизиты реквизита» ссылочного типа очень просто. В окне реквизитов редактора формы документа РасходныйКассовыйОрдер раскроем реквизит Контрагент и перетащим нужные реквизиты (ИНН и Поставщик) справочника Контрагенты. Путь к данным этих «реквизитов реквизита» будет указан через точку – `Объект.Контагент.ИНН`, `Объект.Контагент.Поставщик` (рис. 2.204).

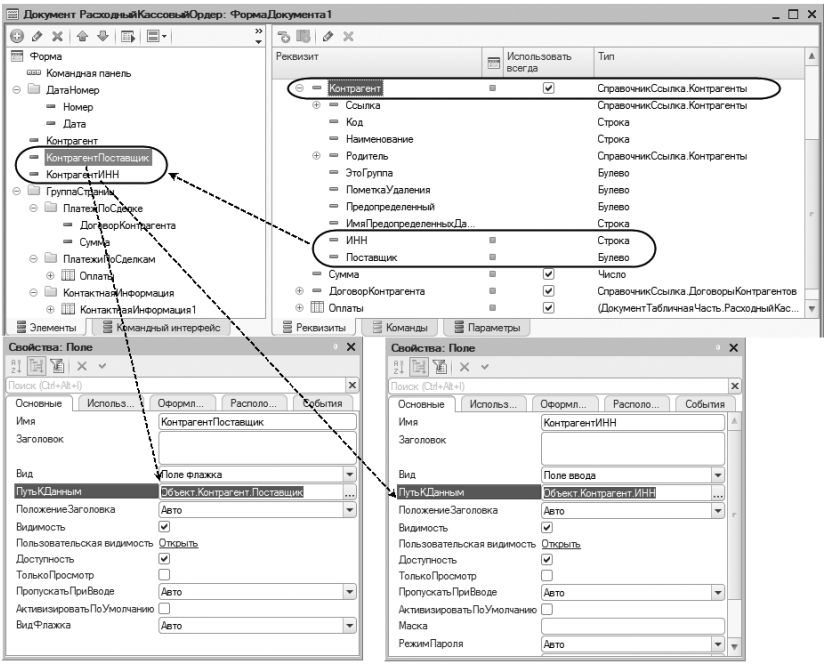


Рис. 2.204. Добавление в форму «реквизитов реквизита»

Теперь добавим в форму обычную группу Дополнительно для просмотра этой дополнительной информации контрагента. Чтобы группа была свертываемой, зададим ее свойства (см. рис. 2.205):

- Заголовок – «Дополнительно <<»;
- Поведение – Свертываемая;
- ЗаголовокСвернутогоОтображения – «Дополнительно >>»;
- Свернута – Да;
- ОтображениеУправления – Гиперссылка заголовка;
- Отображение – Нет.

Перетащим в эту группу элементы формы КонтрагентПоставщик и КонтрагентИНН, а саму группу поместим под полем Контрагент (рис. 2.205).

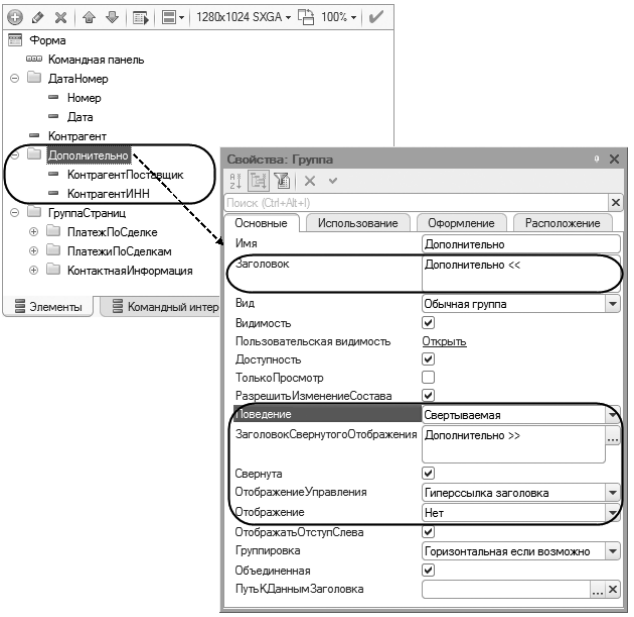


Рис. 2.205. Установка свойств свертываемой группы

В качестве заключительного штриха установим свойство ВидФлажка у поля флажка КонтрагентПоставщик в значение Тумблер.

В результате мы вывели в форму дополнительную информацию о контрагенте и обеспечили возможность ее опционального просмотра (рис. 2.206).

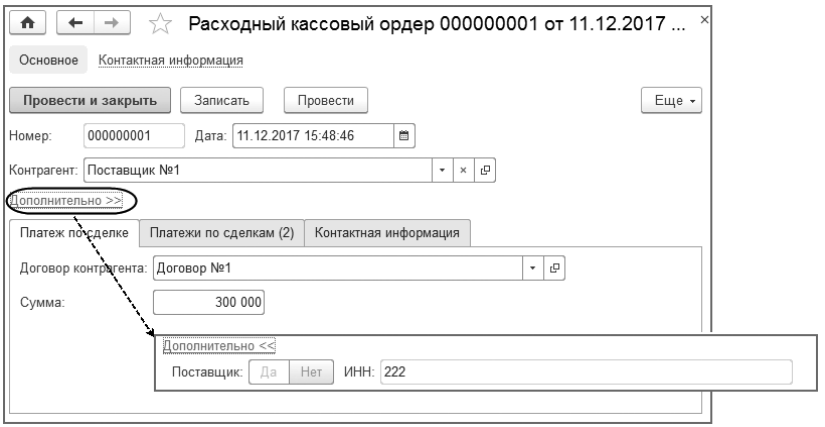


Рис. 2.206. Вывод дополнительной информации в свертываемой группе

Как сгруппировать данные в списке

Часто для большей наглядности и структурированности данных требуется сгруппировать информацию в списке по значению одного или нескольких полей.

Существуют два пути решения поставленной задачи. Первый – в конфигураторе, во время разработки формы. Этот путь следует применять, если подобная группировка будет необходима максимальному количеству пользователей формы. Второй путь – настройка списка в пользовательском режиме работы. Эта настройка будет доступна только тому пользователю, который ее произвел.

Сразу оговоримся, что речь пойдет о настройке динамических списков. Обычные таблицы значений, списки значений подобной настройке не поддаются.

Настроим форму списка документа РасходныйКассовыйОрдер в конфигураторе. Предположим, нам нужно сгруппировать список документов по контрагентам, а внутри них по датам документов.

В редакторе формы списка мы видим, что элемент формы с именем Список отображает данные, имеющие тип Динамический список (рис. 2.207). Это как раз то, что нам необходимо.

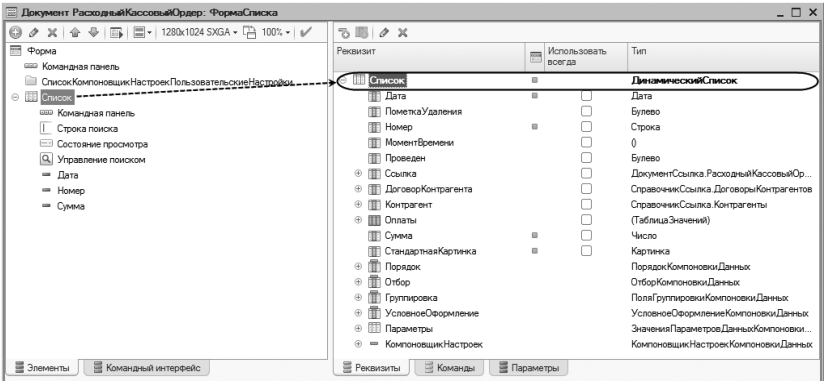


Рис. 2.207. Таблица формы, отображающая данные типа «Динамический список»

Чтобы настроить группировку, необходимо открыть свойства реквизита формы Список и нажать на гиперссылку Открыть свойства Настройка списка (рис. 2.208).

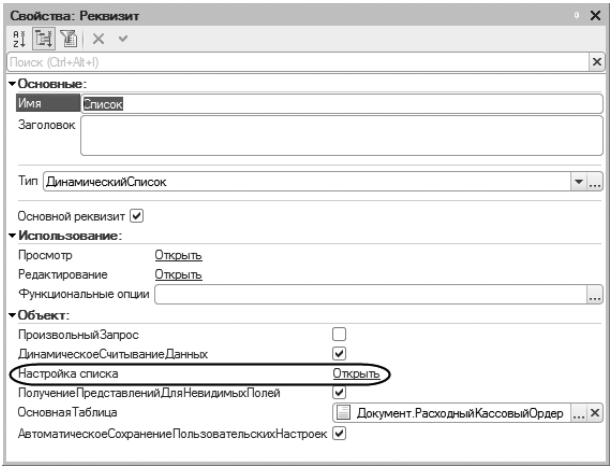


Рис. 2.208. Открытие настроек динамического списка

На закладке Группировка окна Динамический список можно настроить необходимые группировки. Из списка доступных полей перенесем в список полей группировки поля Контрагент и Дата (рис. 2.209).

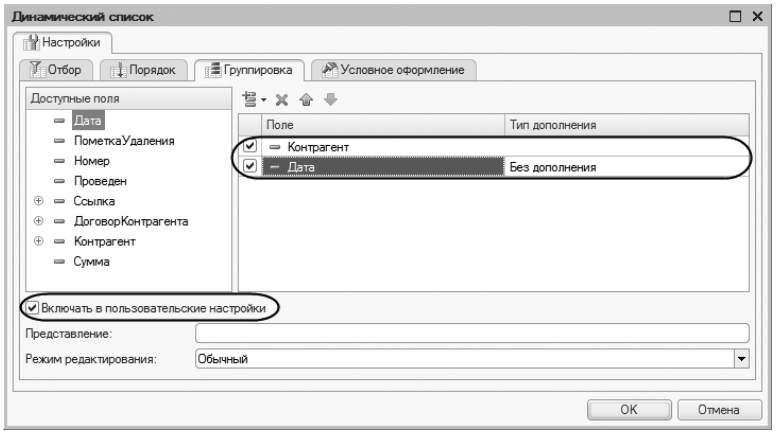


Рис. 2.209. Настройка группировок

В пользовательском режиме список документов будет сгруппирован так, как мы его настроили в конфигураторе. Поскольку флажок Включать в пользовательские настройки стандартно установлен (см. рис. 2.209), то эти настройки будут видны и доступны для изменения по команде Еще – Настроить список (рис. 2.210).

Несмотря на то что Динамический список формируется с помощью системы компоновки данных, получить итоги по группировкам нельзя.

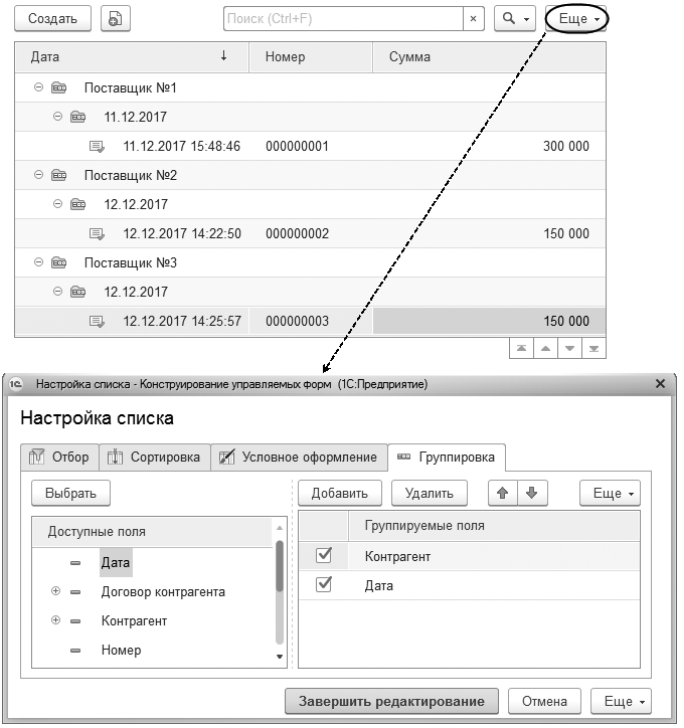


Рис. 2.210. Группировки списка

Как настроить условное оформление динамического списка

Условное оформление динамических списков, так же как и группировки, настраивается в окне настройки динамического списка (см. предыдущий раздел этой главы). Настройки производятся на закладке Условное оформление.

Например, нам требуется выделить сиреневым цветом фона колонку Сумма тех документов, у которых сумма оплаты равна или превышает 200 000. Добавим элемент условного оформления динамического списка (в общем случае таких элементов может быть несколько).

Цвет фона и другие способы оформления данных настраиваются в поле Оформление (рис. 2.211).

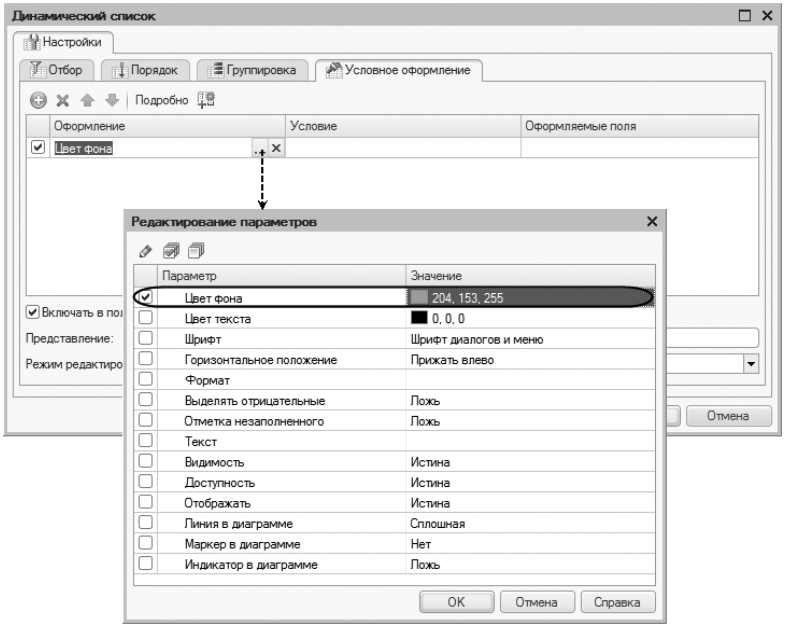


Рис. 2.211. Настройка оформления

Затем в поле Условие зададим условие отбора данных, при котором данные будут оформлены (рис. 2.212).

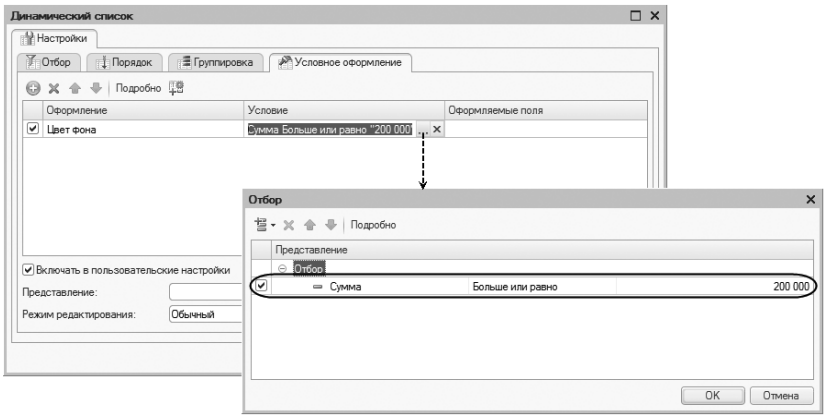


Рис. 2.212. Настройка условия

В поле Оформляемые поля укажем список полей, которые будут выделены при наступлении условия. Если необходимо выделить всю строку, то выбор полей осуществлять не нужно (рис. 2.213).

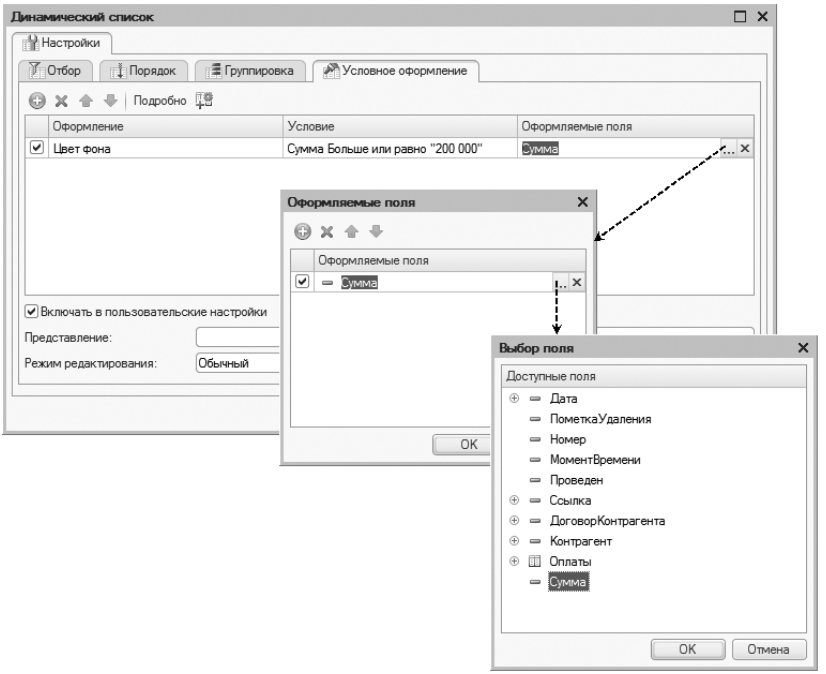


Рис. 2.213. Настройка оформляемых полей

Если известно имя поля, то может быть удобнее не выбирать его из списка доступных полей, а подставлять вручную, набрав первые символы имени поля.

Результат в пользовательском режиме будет выглядеть вот так (рис. 2.214).

Аналогичным образом в конфигурации может быть настроено условное оформление и самой формы. Для этого используется гиперссылка Условное оформление в палитре свойств формы.

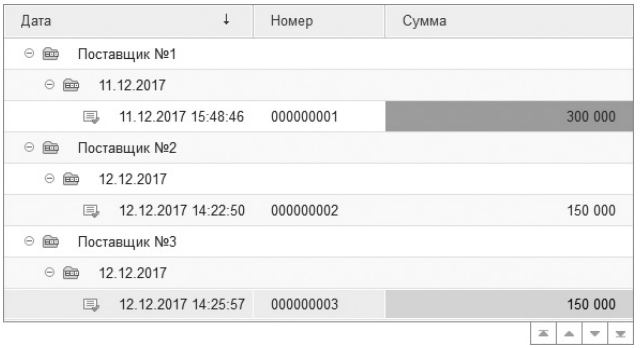


Рис. 2.214. Условное оформление списка

Однако не рекомендуется использовать условное оформление формы для оформления динамических списков, если такое оформление можно получить за счет настройки условного оформления самого списка.

Как усовершенствовать внешний вид формы

При разработке формы помимо собственно ее функциональности очень важны дизайн, лаконичность и интуитивная понятность интерфейса формы. При взгляде на форму пользователь должен (как правило, без чтения инструкций) сразу понимать, для чего предназначена эта форма, что в ней главное, а что второстепенное, и т. д.

Этого можно добиться, в первую очередь, путем различных группировок элементов формы (см. раздел «Как и зачем объединять элементы формы в группы» на стр. 381), расположения их на разных страницах (см. раздел «Как добавить в форму группу страниц» на стр. 391), выделением групп и т. п.

Комбинируя группы с разными настройками группировки и вкладывая группы одна в другую, можно создавать достаточно сложные (по структуре) формы. При этом важное значение имеет выравнивание элементов формы как внутри групп, так и между ними. Потому что форма с невыровненными элементами выглядит неаккуратно, от нее «рябит в глазах», и вследствие этого интерфейс формы кажется сложным и непонятным.

Разработчик может влиять на размещение элементов с помощью следующих параметров:

- горизонтальный и вертикальный интервал размещения элементов;
- размер элементов;
- выравнивание элементов;
- возможность растягивать элементы по горизонтали и вертикали;
- и т. д.

Платформа содержит множество механизмов для автоматического определения размеров и размещения элементов формы. Такая авторасстановка позволит форме выглядеть достаточно хорошо практически на любых устройствах, при любом разрешении и ориентации экрана.

Поэтому при разработке формы следует:

- По возможности использовать значения по умолчанию для свойств элементов. В большинстве случаев должны быть установлены свойства:
  - АвтоМаксимальнаяШирина (для реквизитов),
  - АвтоМаксимальнаяВысота (для реквизитов),
  - Объединенная (для групп),
  - СквозноеВыравнивание (для групп),
  - и т. д.
- Стараться избегать явного указания размеров элементов формы. Свойства:
  - Ширина,
  - Высота.
- Не использовать декорации-пустышки для растягивания и выравнивания элементов формы, кроме создания горизонтального отступа для подчиненных флажков или других элементов формы.

С рекомендациями и примерами адаптации уже существующих интерфейсов «1С:Предприятия» со снятым режимом совместимости к современным версиям платформы можно познакомиться в статье, доступной по ссылке https://its.1c.ru/db/metod8dev#content:5898:hdoc.

Ниже мы рассмотрим несколько примеров, разработанных на новых конфигурациях, поясняющих вышесказанные рекомендации.

Выравнивание между группами

Пусть в конфигурации существует форма констант ОбщиеНастройки, в которой содержатся и константы, относящиеся к настройкам предприятия, и константы, определяющие настройки отдельной организации.

Чтобы зрительно разделить эти группы настроек на два вертикальных столбца, в форму добавлена родительская группа (обычная группа без отображения) Шапка с горизонтальной группировкой. И в нее вложены две дочерние обычные группы с вертикальной группировкой: Левая с заголовком «Настройки предприятия» и Правая с заголовком «Настройки организации». Константы распределены между ними соответственно своему прикладному смыслу (рис. 2.215).

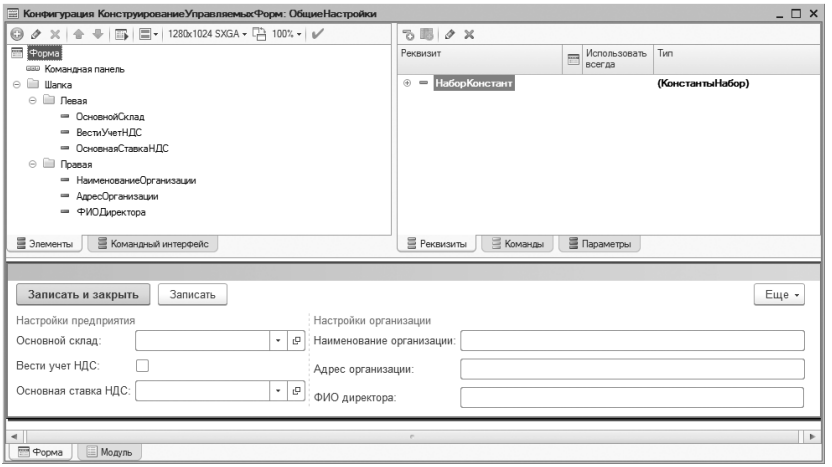


Рис. 2.215. Форма «Общие настройки» в редакторе формы

В режиме 1С:Предприятие настройки разделены на два вертикальных столбца, как и задумывалось. Но если увеличить масштаб формы и внимательно приглядеться, то можно заметить, что строки левого и правого столбца настроек не выровнены между собой по горизонтали (рис. 2.216).

Рис. 2.216. Форма «Общие настройки» в режиме «1С:Предприятие»

Это несоответствие начинается со второй строки настроек. Проблема в том, что в левом столбце второй строки расположена настройка в виде поля флажка, а в правом столбце – в виде поля ввода. Высота элементов формы определяется платформой автоматически, как и рекомендовано, но при этом высота флажка меньше высоты поля ввода. Это оказывает отрицательное влияние на общее выравнивание элементов между группами.

Но решение данной проблемы есть.

Дело в том, что все новые группы стандартно создаются с установленным свойством Объединенная. С помощью этого свойства можно создавать группы, состоящие из нескольких колонок и строк одновременно, с поддержкой выравнивания элементов во всех колонках и строках:

- свойство установлено – группа считается единым элементом;
- свойство сброшено – каждый элемент, входящий в состав группы со сброшенным свойством, считается отдельным элементом.

То есть нам как раз нужен второй вариант. Мы должны снять флажок Объединенная у обеих групп настроек: Левая и Правая. В результате группы перестанут считаться единым элементом и выравнивание элементов по строкам между группами будет восстановлено (рис. 2.217).

Во всех похожих случаях, когда в группах присутствуют разнотипные элементы (например, поле ввода и декорация), нужно поступать аналогичным образом.

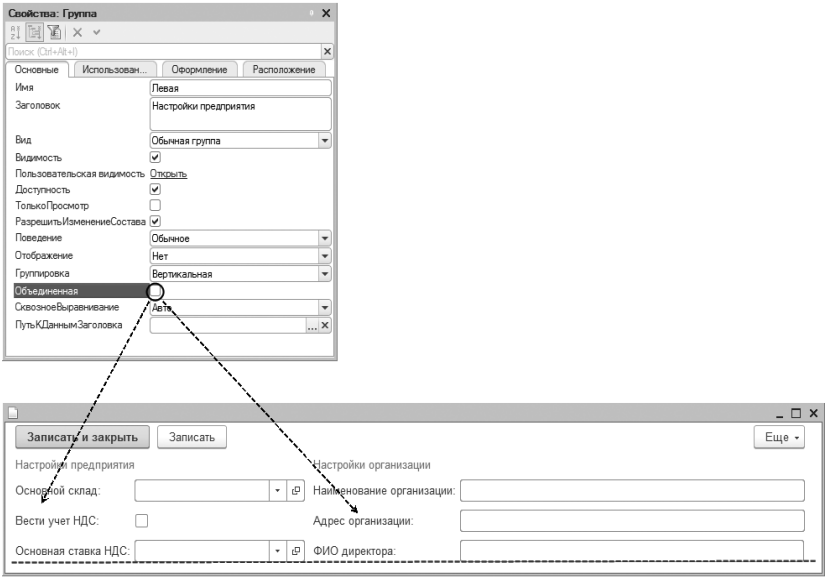


Рис. 2.217. Проверка формы «Общие настройки» из редактора формы по кнопке «Проверить»

Кроме того, при снятии флажка Объединенная между группами не формируются вертикальные разделители, потянув за которые мышью, пользователь может интерактивно изменить ширину вертикального столбца. Поэтому, если эти разделители не нужны или мешают, также нужно снять флажок Объединенная с вертикальных вложенных групп, даже если с выравниванием элементов нет проблем.

Относительное расположение элементов группы

Продолжим наш пример. Предположим, в форме настроек несколько страниц, и мы хотим расположить в форме кнопки для перехода по страницам, прижав кнопку Назад к левому краю, а кнопку Вперед к правому краю формы.

Этого можно добиться следующим образом. Добавим в форму обычную группу без отображения Команды с горизонтальной группировкой. Вложим в нее две дочерние обычные группы без отображения с горизонтальной группировкой: Левые и Правые. Из окна команд

перетащим каждую команду в соответствующую ей группу. Чтобы элементы из группы команд Правые были выровнены по правому краю формы, установим свойства этой группы (рис. 2.218):

- ГоризонтальноеПоложениеПодчиненных – Право;
- РастягиватьПоГоризонтали – Да.

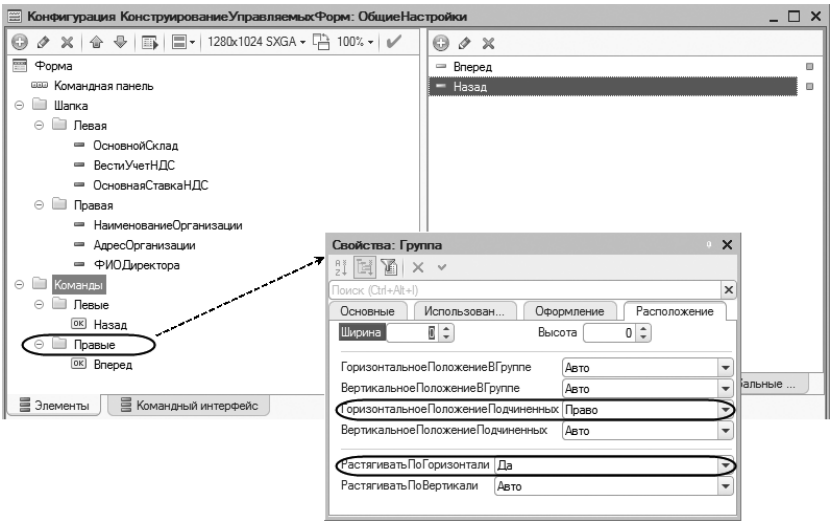


Рис. 2.218. Форма «Общие настройки» в редакторе формы

Проверим, как будет выглядеть форма настроек, нажав кнопку Проверить в командной панели редактора формы (рис. 2.219).

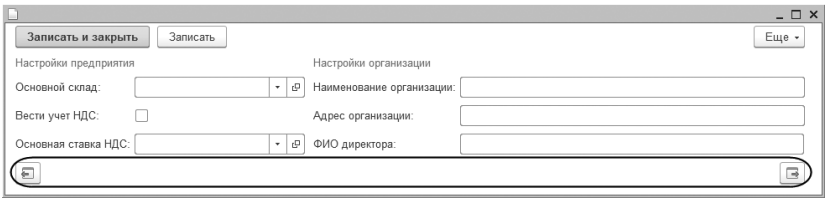


Рис. 2.219. Проверка формы «Общие настройки» из редактора формы по кнопке «Проверить»

Аналогичного результата можно добиться, если вложить в родительскую группу две командные панели. И установить у правой панели свойство ГоризонтальноеПоложение в значение Право. Но этим способом можно пользоваться только для размещения кнопок. В то время как показанным выше способом можно пользоваться для любых элементов формы, входящих в обычную группу.

Создание горизонтального отступа с помощью декораций

При разработке форм не рекомендуется использовать декорации-пустышки для растягивания и выравнивания элементов формы. Один из редких случаев, когда применение таких декораций необходимо – горизонтальный отступ подчиненных флажков (или других элементов). Других инструментов для оформления таких отступов нет.

Например, в форму настроек требуется вывести несколько флажков, одни из которых подчинены другим. Чтобы сразу было понятно, какие флажки главные, а какие зависимые, нужно выводить подчиненные флажки с некоторым горизонтальным отступом от главных. Сделать это можно с помощью структуры вложенных групп и пустых декораций.

Предположим, в нашей форме настроек есть флажок ВестиУчетНДС, от которого зависят флажок ВыводитьНДС и поле переключателя ОсновнаяСтавкаНДС. Чтобы показать эту зависимость, сразу под полем флажка ВестиУчетНДС добавим обычную группу без отображения Детали с горизонтальной группировкой. Добавим в эту группу декорацию-надпись Отступ, которая и обеспечит горизонтальный отступ зависимых настроек. Зададим рекомендованную ширину декорации – 2 (см. рис. 2.220).

Затем в группу Детали под декорацией добавим группу ДеталиНДС с вертикальной группировкой, в которой и будут находиться зависимые элементы: флажок ВыводитьНДС и поле переключателя ОсновнаяСтавкаНДС. У флажков ВестиУчетНДС и ВыводитьНДС установим свойство ПоложениеЗаголовка в значение Право, а у флажка ВыводитьНДС покажем еще расширенную подсказку

и зададим ее заголовок. Свойство ОтображениеПодсказки у флажка установим в значение Отображать снизу (рис. 2.220).

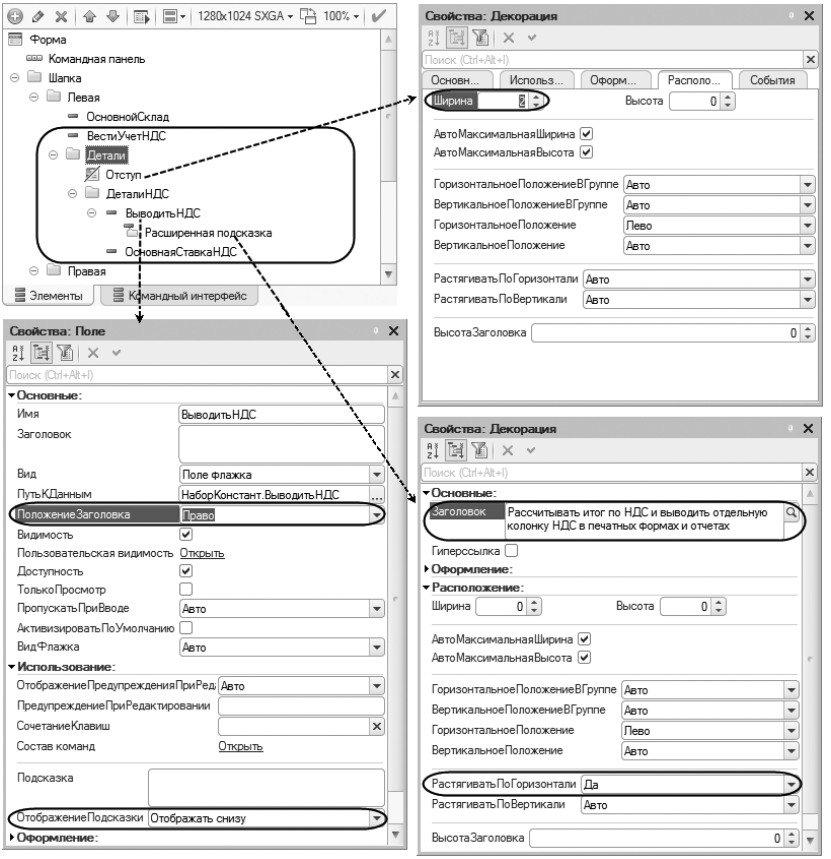


Рис. 2.220. Форма «Общие настройки» в редакторе формы

Обратите внимание, что мы установили у расширенной подсказки свойство РастягиватьПоГоризонтали в значение Да. Это рекомендованное значение, чтобы не было проблем с выравниванием элементов в соседних вертикальных столбцах. Хотя в данном случае можно было бы этого и не делать, т. к. проблем с выравниванием не было и без растягивания.

При нажатии кнопки Проверить в редакторе формы форма настроек будет выглядеть следующим образом (рис. 2.221).

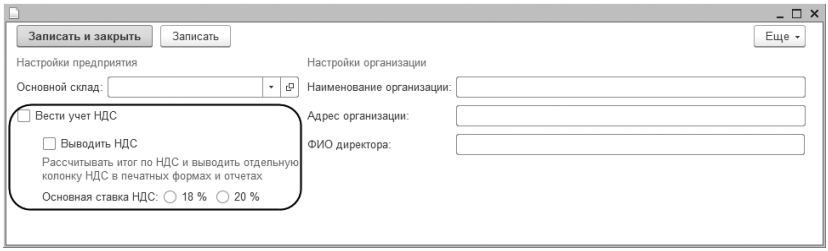


Рис. 2.221. Проверка формы «Общие настройки» из редактора формы по кнопке «Проверить»

Это, конечно, дело вкуса, но структура подчиненности флажков лучше смотрится, когда заголовки флажков располагаются справа.

Глава 2.10. Начальная страница

Начальная страница не является формой в прямом смысле этого слова. Можно сказать, что начальная страница строится разработчиком прикладного решения из уже имеющихся в конфигурации форм и команд. При этом форма обязательно должна быть создана и включена в состав конфигурации. Использовать автоматически генерируемые формы начальная страница не умеет. В прикладном решении начальная страница представлена в единственном экземпляре.

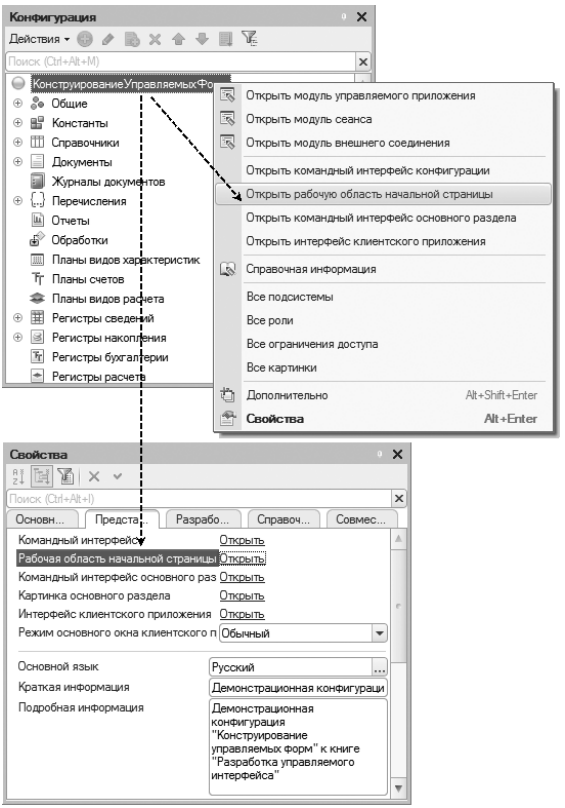


Рис. 2.222. Открытие начальной страницы в режиме «Конфигуратор»

Настроить начальную страницу при разработке прикладного решения можно через свойства разрабатываемой конфигурации или контекстное меню (рис. 2.222).

Создание и редактирование начальной страницы происходят в специализированном редакторе (рис. 2.223).

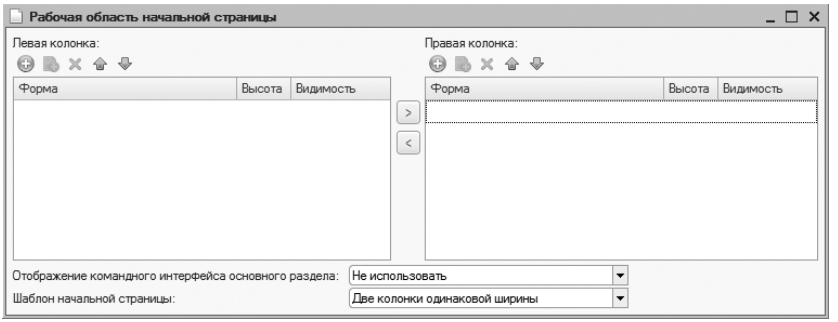


Рис. 2.223. Редактор «Рабочая область начальной страницы»

Формы, отображаемые на начальной странице в пользовательском режиме работы, могут располагаться в одну или две колонки. Настройка количества колонок с формами осуществляется с помощью выбора одного из предопределенных значений поля Шаблон начальной страницы.

При этом доступны следующие значения:

- Одна колонка. Формы будут располагаться вертикально, одна под другой (рис. 2.224). Управлять последовательностью форм можно с помощью кнопок Переместить вверх, Переместить вниз.

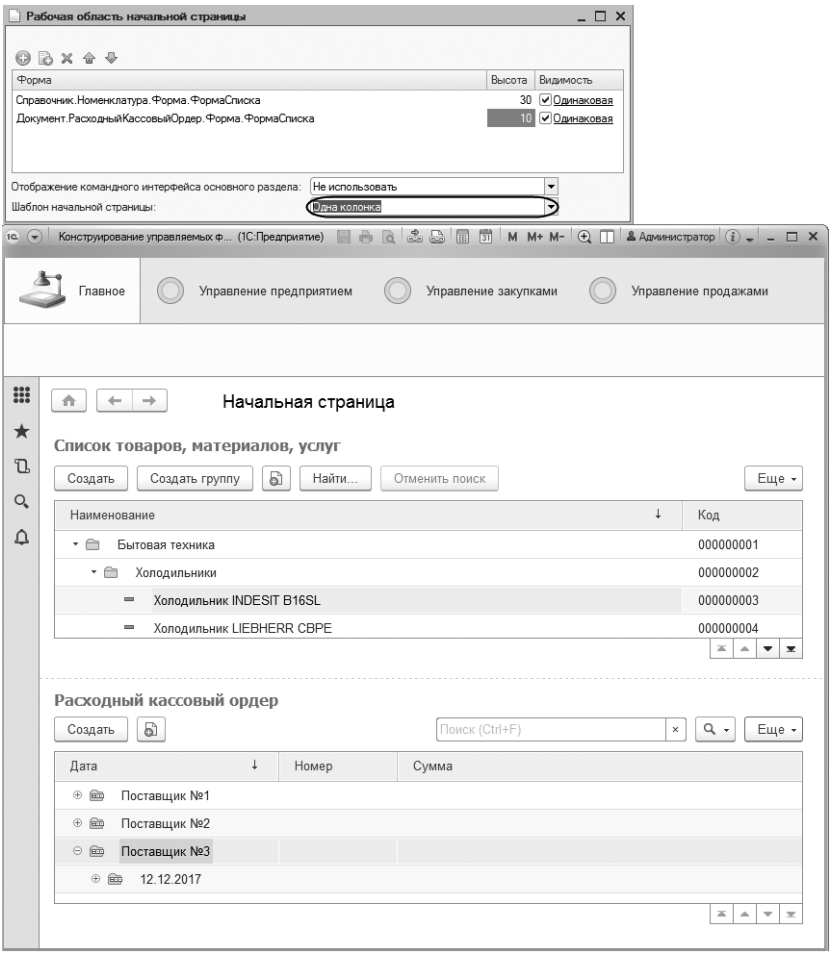


Рис. 2.224. Шаблон «Одна колонка»

- Две колонки одинаковой ширины. Формы будут располагаться в две колонки. Переносить форму из одной колонки в другую можно с помощью кнопок, расположенных между колонками редактора начальной страницы. Горизонтальный размер форм в разных колонках будет одинаковым (рис. 2.225).

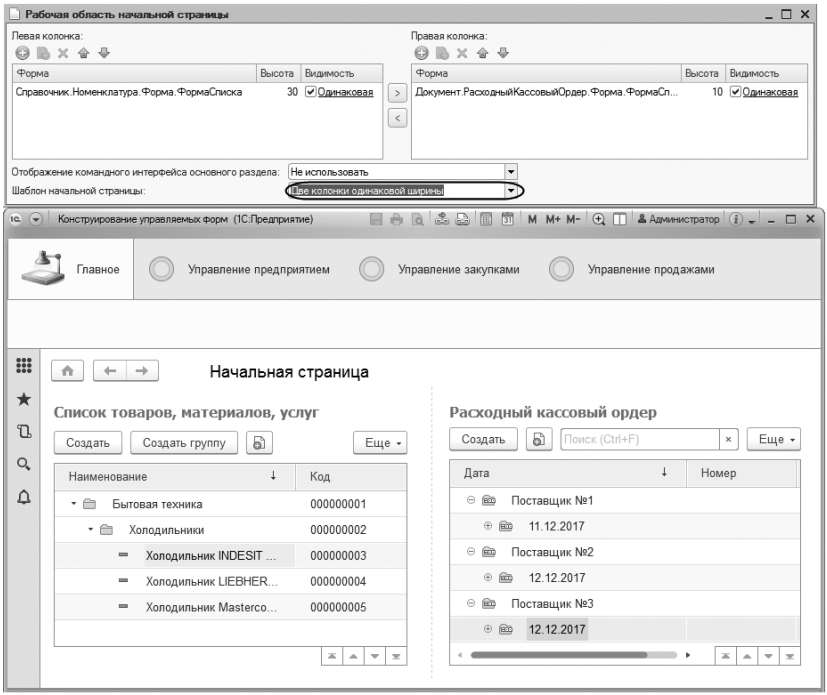


Рис. 2.225. Шаблон «Две колонки одинаковой ширины»

- Две колонки разной ширины (2:1). Формы будут располагаться, как и в предыдущем примере, однако ширина форм первой колонки будет в два раза больше ширины форм второй колонки начальной страницы (рис. 2.226).

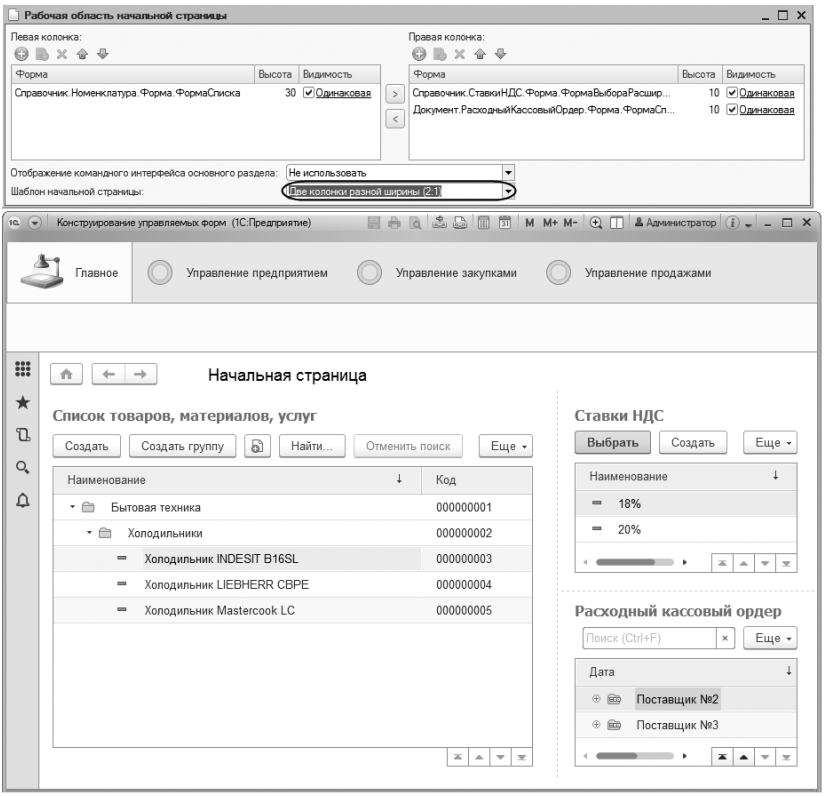


Рис. 2.226. Шаблон «Две колонки разной ширины»

Добавление форм в состав начальной страницы производится стандартным способом – с помощью кнопки командной панели. При этом разработчику необходимо выбрать конкретную форму, указав ее в списке (рис. 2.227).

Кроме управления шириной форм начальной страницы с помощью ширины колонок возможно управление их высотой. Высота форм задается в строках, и для каждой из форм она своя. Если видимая часть, отведенная форме на начальной странице, меньше, чем высота формы, то у последней появляется вертикальная полоса прокрутки (рис. 2.228).

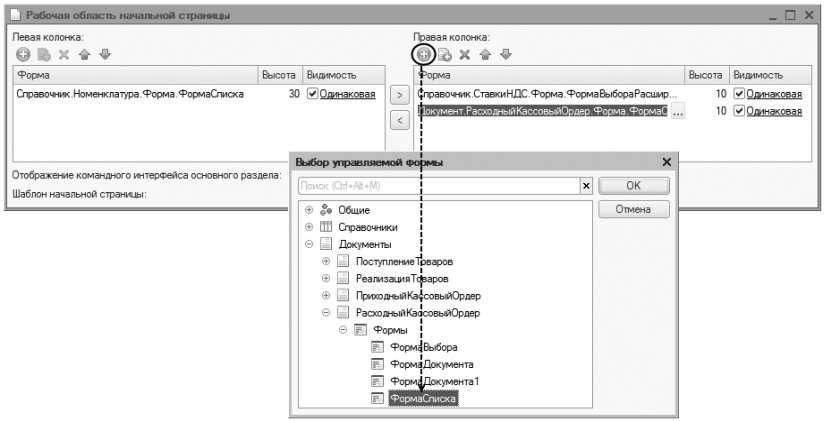


Рис. 2.227. Добавление формы на начальную страницу

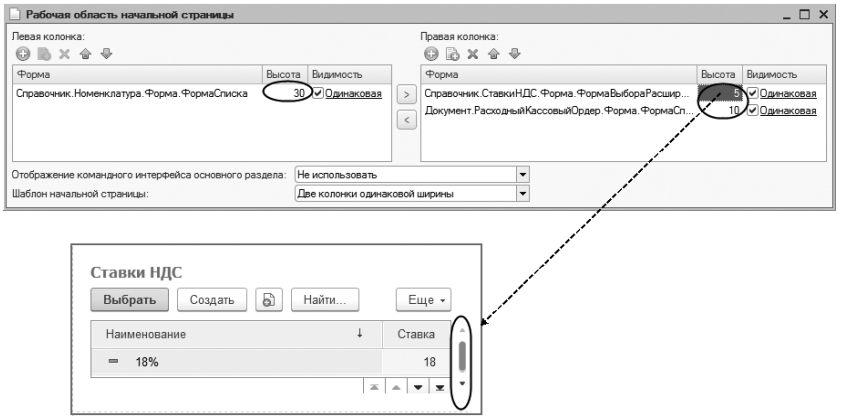


Рис. 2.228. Задание высоты форм

Разработчик прикладного решения может влиять на состав форм начальной страницы, отображаемой в пользовательском режиме работы, с помощью настройки видимости форм. Например, если снять общую видимость у формы списка документа Расходный кассовый ордер, то эта форма не будет отображаться на начальной странице для любого из пользователей прикладного решения (рис. 2.229).

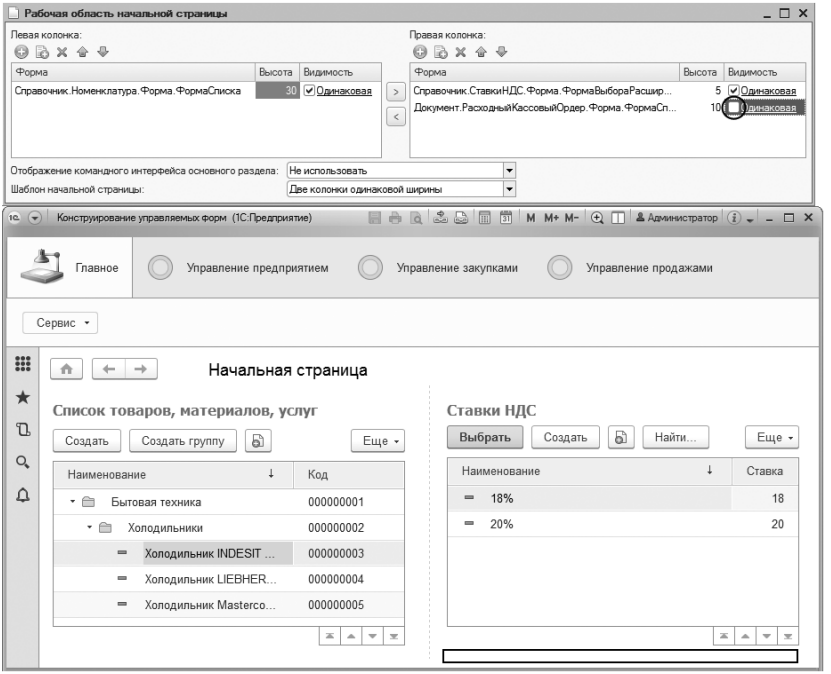


Рис. 2.229. Настройка видимости форм на начальной странице

Настройка видимости форм является особенно важной для тех категорий пользователей, которые обладают наиболее полными правами на объекты конфигурации или совмещают несколько ролей прикладного решения. Только таким образом можно будет избежать «каши» на начальной странице такого пользователя.

Состав форм начальной страницы приложения зависит от прав доступа, которыми обладает пользователь прикладного решения. Так, если совокупность ролей пользователя не дает прав на просмотр некой формы объекта конфигурации, то такая форма не будет отображаться на начальной странице. Например, у роли Полные права установлена видимость формы списка документа Расходный кассовый ордер, а у роли Менеджер по продажам – нет (рис. 2.230).

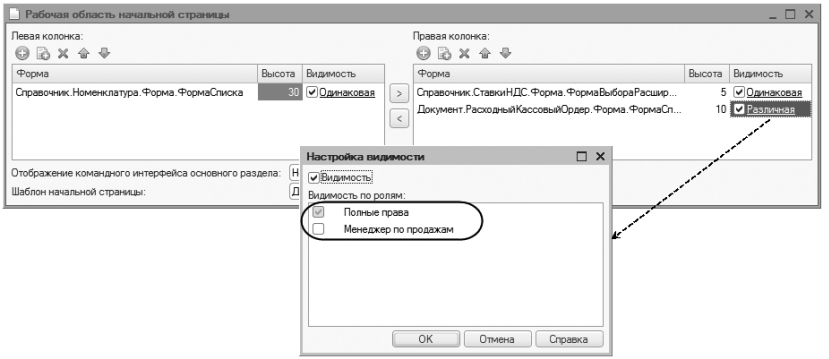


Рис. 2.230. Установка видимости по ролям для форм начальной страницы

В результате на начальной странице пользователя с ролью Полные права будет видна форма списка документа Расходный кассовый ордер, а на начальной странице пользователя с ролью Менеджер по продажам – нет (рис. 2.231, 2.232).

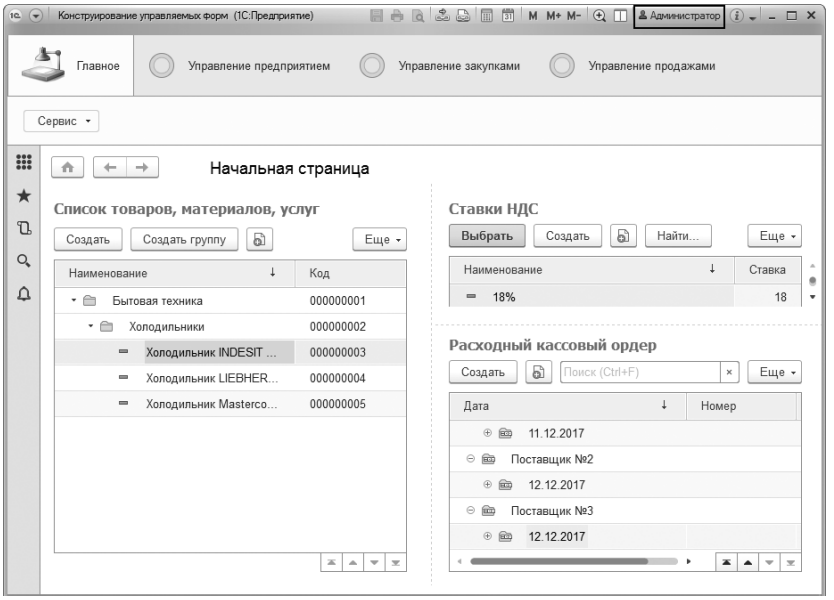


Рис. 2.231. Начальная страница пользователя с ролью «Полные права»

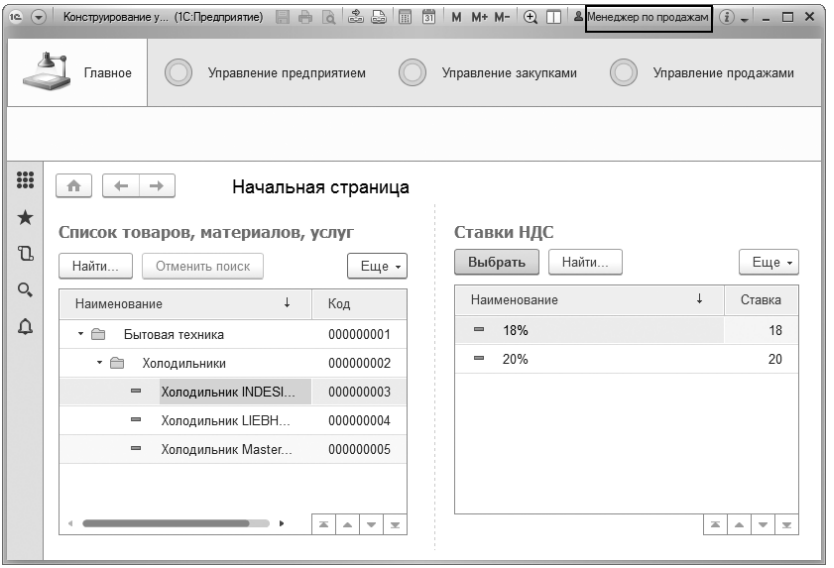


Рис. 2.232. Начальная страница пользователя с ролью «Менеджер по продажам»

Функциональные опции, реализованные в прикладном решении, также оказывают влияние на отображение форм, находящихся на начальной странице пользователя. Например, если функциональная опция Вести учет НДС отключена, то форма списка справочника СтавкиНДС, который от нее зависит, не будет показана на начальной странице (рис. 2.233).

Пользователь исходя из своих предпочтений также может изменить состав форм на начальной странице из списка доступных форм, предназначенных для этого разработчиком. Для этого нужно выполнить команду главного меню Вид – Настройка начальной страницы ... (рис. 2.234).

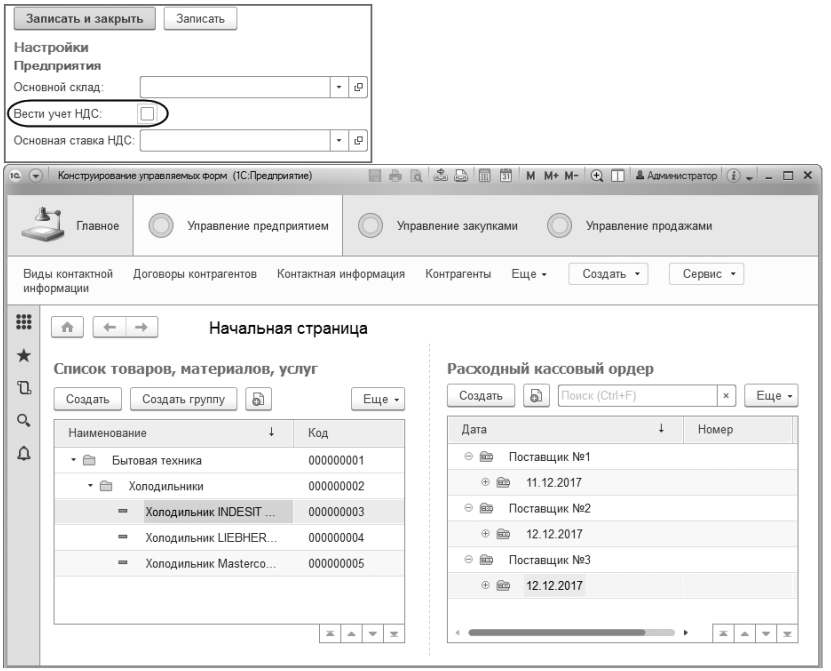


Рис. 2.233. Влияние функциональных опций на состав форм начальной страницы

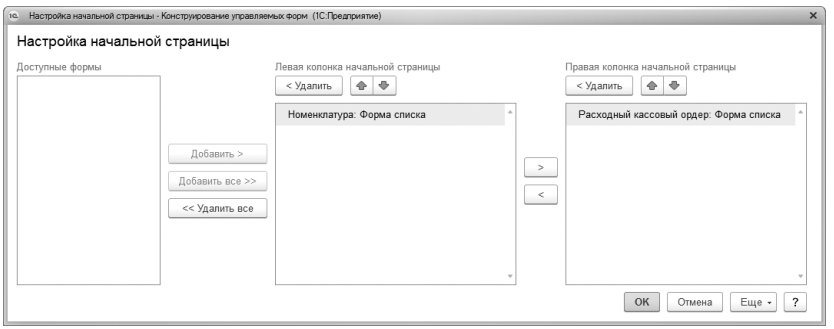


Рис. 2.234. Пользовательская настройка начальной страницы

Чтобы дать «свободу действий» пользователю при настройке начальной страницы в режиме 1С:Предприятие, разработчик может поместить на начальную страницу в конфигураторе те формы,

которые, по его мнению, не важны, но могут понадобиться пользователю. Если снять общую видимость с этих форм, то они появятся в списке доступных форм и будут доступны при настройке в пользовательском режиме.

Кроме того, состав и расположение форм на начальной странице можно настроить программно при старте прикладного решения. Об этом будет рассказано в третьей части книги, в разделе «Настройка состава форм на начальной странице» на стр. 731.

ЧАСТЬ 3

Программирование форм и интерфейса

Оглавление

Глава 3.1. Форма как элемент клиент-серверного взаимодействия

Прежде чем рассматривать конкретные примеры программирования в формах, нужно иметь четкое представление о том, что представляет собой форма. В предыдущих главах книги уже давались общие сведения о ее устройстве и функционировании, сейчас же мы рассмотрим форму более подробно и именно с точки зрения ее внутреннего, «программного» устройства.

Клиент-серверная архитектура

Форма – это программный объект, который создается в процессе работы прикладного решения в режиме 1С:Предприятие.

Форма может быть создана платформой без участия разработчика. Либо она может быть создана по инициативе разработчика в результате выполнения алгоритма, описанного разработчиком на встроенном языке.

В «1С:Предприятии» нет «единого пространства» для выполнения кода на встроенном языке. Каждый фрагмент кода выполняется в некотором *контексте*.

По большому счету, не углубляясь в детали, таких контекстов два: это *контекст клиента* и *контекст сервера*.

С одной стороны, контекст определяет «программное окружение», в котором исполняется код: доступные свойства, методы, набор объектов встроенного языка, которые можно использовать.

С другой стороны, контекст определяет физическое место (конкретный компьютер), на котором выполняется этот код.

Например, в клиент-серверном варианте работы «1С:Предприятия» взаимодействие программных компонентов будет выглядеть следующим образом (рис. 3.1).

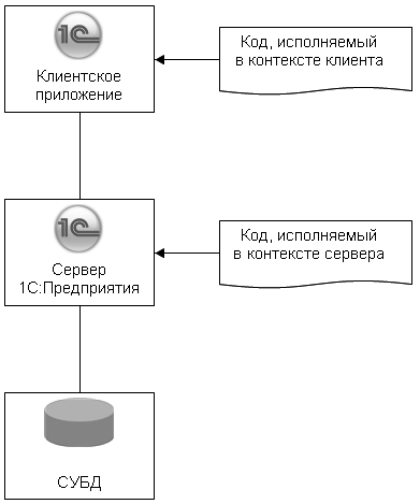


Рис. 3.1. Программные компоненты «1С:Предприятия» в клиент-серверном варианте работы

Клиентское приложение – это программа (толстый клиент, тонкий клиент или веб-клиент), которая обеспечивает интерактивное взаимодействие системы с пользователем. Она может помимо всего прочего исполнять код на встроенном языке, написанный разработчиком. В этом случае говорят, что этот код исполняется *в контексте клиентского приложения*, или, короче, *на клиенте*.

Сервер «1С:Предприятия» – это один из рабочих процессов, функционирующих в составе кластера серверов «1С:Предприятия». Он обеспечивает взаимодействие клиентского приложения с хранилищем данных – системой управления базами данных (СУБД). А кроме этого сервер «1С:Предприятия» также может исполнять код на встроенном языке. В этом случае говорят, что код исполняется *в контексте сервера*, или, короче, *на сервере*.

Первая особенность, которая важна для нас, заключается в том, что на сервере и на клиенте доступны разные свойства, методы и объекты встроенного языка. Грубо говоря, все действия, связанные с доступом к данным СУБД (их чтение и запись), возможны только на сервере, в то время как интерактивные действия (отображение данных поль-

зователю, визуальное взаимодействие с пользователем) возможны только на клиенте.

Теперь обратим внимание на другую особенность, которая также будет важна для нас. Программные компоненты «1С:Предприятия», о которых мы говорили выше, могут размещаться как на одном компьютере, так и на разных. Вариантов их размещения существует много, но наиболее реальной ситуацией «боевой» работы клиент-серверного варианта будет ситуация, показанная на рисунке 3.2.

Рис. 3.2. Схема клиент-серверного варианта работы

Клиентское приложение работает на компьютере пользователя – *клиентском компьютере*. Сервер «1С:Предприятия» работает на отдельном выделенном компьютере – *сервере*. СУБД также работает на отдельном компьютере – *сервере базы данных*.

Как правило, сервер и сервер баз данных находятся в пределах одной локальной сети. А вот клиентский компьютер, вообще говоря, может находиться где угодно. Он может находиться в этой же сети, а может подключаться к серверу через Интернет. При этом каналы

связи, используемые для такого подключения, могут быть самыми разными – от широкого и быстрого оптоволоконного подключения до узкого и медленного подключения по каналам сотовой связи.

Работа системы «1С:Предприятие» всегда начинается на компьютере пользователя – пользователь запускает приложение и соединяется с информационной базой. Уже в этот момент в работу вступает серверная часть «1С:Предприятия» – выполняется соединение с кластером серверов, аутентификация подключающегося пользователя в информационной базе, формирование основного окна системы, начальной страницы и передача его назад, на клиент. В дальнейшем этот процесс повторяется неоднократно: выполнение каких-либо действий начинается в клиентском приложении, затем управление передается на сервер, он выполняет некоторые действия и возвращает управление на клиент. То есть клиентское приложение вызывает сервер для выполнения каких-либо действий, сервер выполняет их и возвращает управление и результат этих действий обратно клиентскому приложению.

Механизмы платформы, отвечающие за это взаимодействие, оптимизированы для того, чтобы обеспечивать приемлемую скорость работы системы даже на низкоскоростных каналах связи.

Однако кроме платформы такое взаимодействие может вызвать и сам разработчик в результате работы своего алгоритма на встроенном языке. Например, когда из процедуры, исполняющейся на клиенте, он вызывает процедуру, которая должна исполняться на сервере. В этом случае также происходит вызов сервера, управление передается на него, выполняется требуемая процедура, и управление возвращается обратно на клиент, в ту процедуру, из которой был осуществлен вызов.

Поскольку заранее неизвестно, каким именно образом клиент подключен к серверу, такой серверный вызов может быть «совсем незаметным» для системы, или, наоборот, он может быть очень «затратным». Например, в том случае, когда используются мобильные каналы связи. Поэтому к вызовам сервера разработчик должен относиться очень внимательно. В противном случае быстродействие и производительность разрабатываемой системы могут оказаться очень низкими.

Форма – клиент-серверный объект

Теперь посмотрим, почему эти две особенности так важны именно при программировании форм.

Как мы сказали в самом начале, форма – это программный объект, существующий в памяти компьютера в какой-то момент работы системы в режиме 1С:Предприятие.

Раз мы говорим о его «программировании», то это значит, что этот программный объект создается платформой на основе того описания, которое существует в дереве объектов конфигурации, то есть на основе объекта конфигурации Форма (рис. 3.3). Потому что в «1С:Предприятии» могут существовать формы, не описанные в конфигурации, – те, которые платформа генерирует автоматически. Но в такие формы разработчик не может добавить свой код.

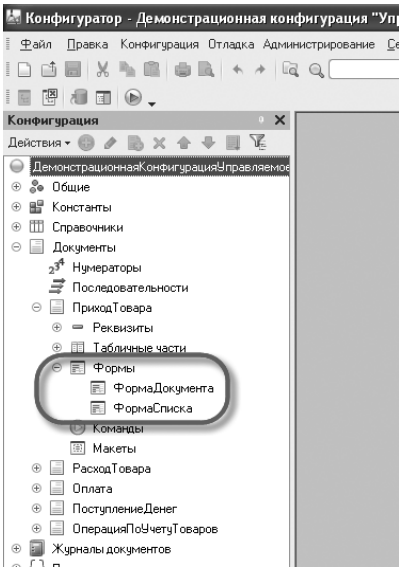


Рис. 3.3. Формы, созданные разработчиком

Каждый такой объект конфигурации Форма, описанный в дереве объектов конфигурации, имеет *модуль*, в котором разработчик располагает собственные процедуры на встроенном языке (рис. 3.4).

Эти процедуры вызываются в определенные, заранее известные моменты работы формы (программного объекта) и определяют ее нестандартное, отличное от других форм, поведение и возможности.

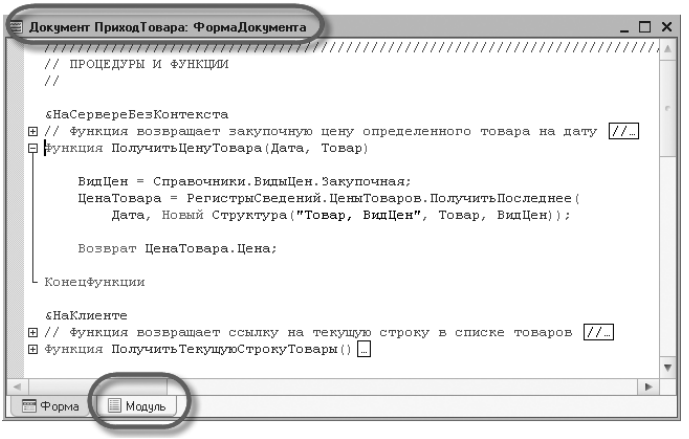


Рис. 3.4. Модуль формы

Так вот, основная особенность формы как программного объекта заключается в том, что она существует и на клиенте, и на сервере.

Соответственно, модуль формы, процедуры, написанные в этом модуле, исполняются не в каком-то одном определенном контексте (только в контексте клиента или только в контексте сервера). Для каждой процедуры в модуле формы разработчик в явном виде указывает контекст ее исполнения: на сервере или на клиенте.

Общий подход к программированию форм

Из этого следуют несколько важных тезисов, определяющих общий подход к программированию форм.

1. Разработчик должен понимать, что он не просто кодирует некий прикладной алгоритм, а в явном виде программирует отдельно клиентскую и отдельно серверную части приложения.
2. Разработчик должен управлять частотой вызовов сервера и объемом передаваемой информации.

3. В разрабатываемой конфигурации код, реализующий бизнес-логику, должен быть четко отделен от кода, реализующего интерфейс.
4. Структура кода должна определяться не прикладной логикой решаемой задачи, а логикой клиент-серверного взаимодействия.
5. Клиентский код пишется не как последовательность действий, которую нужно выполнить. Прежде всего он продумывается как сценарий передачи управления с клиента на сервер и обратно.
6. Несмотря на то что форма существует и на клиенте, и на сервере, клиент и сервер нужно рассматривать не как единое пространство выполнения приложения, а скорее как два взаимодействующих приложения.
7. Полезно мысленно представить вызов сервера как «непростой процесс»:
   - Система формирует обращение к серверу, передает его по каналу связи, потом выполняет его на сервере, возвращает ответ по каналу связи…
   - Если тонкий клиент работает, например, через GPRS, то каждый вызов – это примерно 1,5 секунды!
   - Хочется, чтобы с прикладным решением работали удаленные пользователи? Тогда нужно думать о каждом вызове сервера!

На протяжении последующих глав, особенно в четвертой части книги, эти рекомендации будут неоднократно повторяться и объясняться более подробно. Сейчас же важно усвоить следующее: программирование форм – это отдельное непростое занятие, требующее взвешенного и методически грамотного использования имеющихся возможностей платформы.

Глава 3.2. Параметры и реквизиты формы

Вы уже знаете, что форма, как правило, содержит некоторое количество *реквизитов* и некоторое количество *параметров*.

И те и другие позволяют хранить некоторые данные, доступны в редакторе формы на соответствующих закладках, и те и другие разработчик может добавить самостоятельно, по своему желанию. Для этого в редакторе формы существуют две закладки: Реквизиты и Параметры (рис. 3.5).

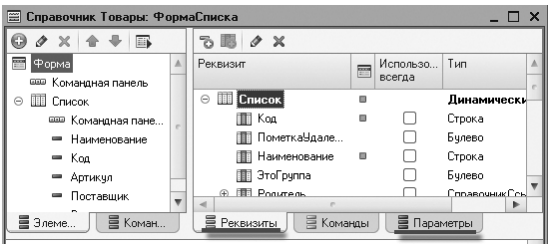


Рис. 3.5. Реквизиты и параметры формы

В чем же различие между параметрами и реквизитами формы? Для каких задач нужно использовать реквизиты, а для каких – параметры?

Реквизиты

Реквизиты формы предназначены для хранения данных, с которыми работает форма. В реквизитах хранятся те данные, которые отображаются и редактируются в элементах формы, а также те данные, которые не отображаются, но используются формой в процессе ее работы. Вся совокупность данных, хранящихся в реквизитах формы, называется *данные формы*. Этот термин мы будем использовать в дальнейшем.

Каждый *элемент формы*, позволяющий изменять данные, связан с некоторым реквизитом формы. Когда пользователь изменяет данные в элементе формы, аналогичные изменения происходят и в связанном с ним реквизите. Верно и обратное. Когда реквизит формы изменяется программно, изменяется и значение, отображаемое в связанном с ним элементе формы.

Не все реквизиты формы обязательно должны отображаться в форме. Часть реквизитов может быть не связана с элементами формы, а данные, хранящиеся в таких реквизитах, предназначены не для отображения или интерактивного редактирования, а для обеспечения внутренних алгоритмов работы формы или для программного взаимодействия с этой формой из встроенного языка.

Можно сказать, что реквизиты формы составляют часть программного интерфейса формы, с помощью которого строится взаимодействие с этой формой «извне» – из других форм или из фрагментов программного кода.

Важным моментом является то, что реквизиты формы существуют на протяжении всего времени существования самой формы.

Параметры

Параметры формы, в отличие от реквизитов, предназначены для управления функциональностью формы при ее открытии и, грубо говоря, существуют лишь в момент создания и открытия формы. После создания или открытия формы все параметры, кроме *ключевых параметров*, удаляются, т. к. для дальнейшей работы формы они уже не нужны.

Ключевые параметры – это особый вид параметров, значения которых могут потребоваться и после того, как форма создана. Например, для того чтобы найти именно этот экземпляр формы среди других открытых форм.

Выводы

Итак, подытоживая сказанное, можно отметить, что для хранения, отображения, изменения данных, для взаимодействия с формой извне, для хранения служебных данных нужно использовать реквизиты формы.

Для того чтобы создать и открыть форму в том виде и в том состоянии, в котором хочется, нужно использовать параметры формы.

Следующая глава как раз будет посвящена имеющимся способам открытия форм, и в ней мы будем оперировать различными параметрами формы.

Глава 3.3. Открытие форм

Самый первый, наверное, вопрос, который возникает у разработчиков, начинающих знакомиться с формами: а как открыть форму?

В этой главе мы рассмотрим общий подход и несколько конкретных примеров открытия форм в типичных ситуациях.

Последовательность событий при открытии формы

При открытии формы нового объекта (элемента справочника, документа и т. п.) и при открытии формы существующего объекта возникает различная последовательность событий.

Если открывается форма нового объекта, то сначала происходит начальное заполнение объекта данными (событие Обработка заполнения в модуле объекта, форма которого открывается), а затем вызываются два события формы: сначала на сервере (При создании на сервере) и затем на клиенте (При открытии). Эти два события позволяют подготовить форму к открытию (рис. 3.6).

Рис. 3.6. Последовательность событий при открытии формы нового объекта

Слева показан клиентский контекст, в котором вызываются события формы. Посередине и справа – серверный контекст, в котором вызываются как события формы, так и события самого прикладного объекта, форма которого открывается.

Если открывается форма существующего объекта, то последовательность событий будет иной (рис. 3.7). Сначала на сервере вызываются два события формы. Одно (При чтении на сервере) – чтобы подготовить дополнительные данные, которые зависят от данных объекта. Другое (При создании на сервере) – чтобы максимально подготовить форму к открытию. И, наконец, на клиенте вызывается еще одно событие формы (При открытии), чтобы выполнить действия, связанные с открытием формы, которые на сервере выполнить невозможно.

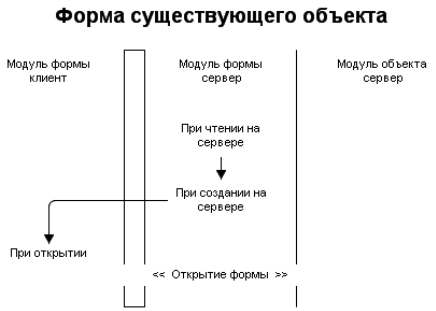


Рис. 3.7. Последовательность событий при открытии формы существующего объекта

Из всех перечисленных событий нас прежде всего будут интересовать два события формы: При создании на сервере и При открытии.

Все, что можно сделать на сервере для подготовки формы к открытию, нужно делать в обработчике события При создании на сервере.

В обработчике события При открытии нужно делать только то, что на сервере сделать невозможно. Например, выдать предупреждение, задать вопрос. Или же выполнить те действия, которые выполняются именно тогда, когда форма наверняка открывается. Например, открыть связанную форму, которая «сама по себе» не существует, а существует, только если открыта родительская форма. Потому что обработчик При открытии – последний перед открытием формы, в котором можно отказаться от ее открытия. И если не происходит отказа от открытия родительской формы, то, значит, она наверняка будет открыта.

Что касается других событий, то, например, о событии Обработка заполнения и механизме начального заполнения объектов мы поговорим подробно в главе 3.10 на стр. 512.

А о событии При чтении на сервере мы поговорим в главе, описывающей преобразование данных объекта в данные формы и обратно, – в главе 3.4 на стр. 471.

Общая методика открытия форм

Общий подход к открытию форм следующий. Для открытия формы используется метод глобального контекста `ОткрытьФорму()`. Он доступен только в контексте клиентов: толстого клиента, тонкого клиента и веб-клиента. На сервере открыть форму нельзя.

Этот метод содержит ряд параметров, которые позволяют указать, какая именно форма должна быть открыта, и задать некоторые свойства открываемой формы.

Значения передаваемых параметров либо применяются платформой автоматически, либо разработчик может применить их самостоятельно в обработчике события формы При создании на сервере.

Рассмотрим наиболее типичные случаи.

Основная форма нового объекта

Открыть основную форму справочника Товары можно следующим образом (листинг 3.1):

Листинг 3.1. Открытие основной формы нового объекта

```
ОткрытьФорму("Справочник.Товары.ФормаОбъекта");
```

Первым параметром метода передается имя той формы, которую нужно открыть. Оно образуется по определенным правилам. Сначала указывается полное имя объекта конфигурации (в данном случае `Справочник.Товары`), а после него, через точку, *стандартное имя основной*

Примеры можно посмотреть в демонстрационной базе «Открытие форм», панель функций текущего раздела, группа команд Открыть основную форму, глобальные команды Объекта, Группы, Списка, Выбора, Выбора группы.

формы (ФормаОбъекта). Все стандартные имена основных форм перечислены в синтакс-помощнике в разделе Глобальный контекст – Процедуры и функции интерактивной работы, в описании метода ОткрытьФорму() (рис. 3.8).

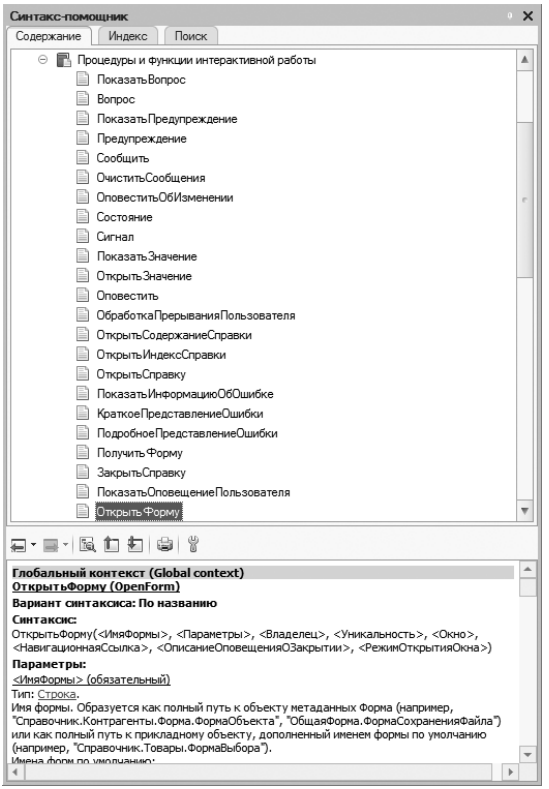


Рис. 3.8. Метод «ОткрытьФорму()» в синтакс-помощнике

Единственной сложностью тут является то, что имя формы указывается как строка, и поэтому писать его нужно внимательно, не забывая о том, что имя класса объектов конфигурации указывается в единственном числе: Справочник, Документ и т. д.

Поскольку кроме имени формы никаких других параметров не указывается, в случае формы объекта будет создан новый объект и открыта его форма.

В случае же форм списка и выбора просто будут открыты соответствующие формы (листинг 3.2).

Как открыть форму существующего объекта, рассказано в разделе «Форма существующего объекта» на стр. 454.

Листинг 3.2. Открытие основных форм списка и выбора

```
ОткрытьФорму("Справочник.Товары.ФормаСписка");
ОткрытьФорму("Справочник.Товары.ФормаВыбора");
ОткрытьФорму("Справочник.Товары.ФормаВыбораГруппы");
```

Таким образом можно открыть как автогенерируемые формы, так и формы, которые созданы разработчиком в конфигурации и назначены основными для объектов конфигурации.

Однако существуют две особенности, связанные с открытием стандартных форм, о которых будет сказано далее.

Форма констант

Первая особенность связана с формой констант. Из встроенного языка нет возможности открыть автогенерируемую форму констант. Можно открыть только форму констант, созданную в конфигурации (листинг 3.3).

Листинг 3.3. Открытие формы констант, созданной в конфигурации

```
ОткрытьФорму("ОбщаяФорма.ФормаКонстант");
```

Здесь `ФормаКонстант` – это имя созданной формы.

Аналогичным образом можно открыть и другие общие формы конфигурации.

Форма группы

Вторая особенность связана с формой группы иерархического справочника или иерархического плана видов характеристик.

Пример можно посмотреть в демонстрационной базе «Открытие форм», панель функций текущего раздела, группа команд Сервис, глобальная команда Открыть форму констант.

Указывая имя формы в методе `ОткрытьФорму()`, мы лишь определяем, какая форма должна быть открыта. Но не определяем особенности элемента данных, который будет отображаться в этой форме.

В случае с документами или линейными справочниками никакой проблемы не возникнет. Все элементы данных в таких структурах равнозначны.

Однако если справочник не линейный, а иерархический, то он содержит как «обычные» элементы, так и элементы, являющиеся группами.

Если при открытии формы не делать специальных указаний, платформа стандартно будет создавать именно новый элемент, а не группу. Поэтому если мы хотим открыть форму новой группы, то помимо указания имени открываемой формы нужно еще указать, что должна создаваться именно новая группа, а не элемент. Для этого используются *параметры формы*.

Некоторый стандартный набор параметров существует у самого объекта встроенного языка `Форма`. В режиме 1С:Предприятие к этим параметрам, которые существуют у любой формы, добавляется еще некоторое количество параметров, поставляемых *расширением формы*.

Параметры формы описаны в синтакс-помощнике (рис. 3.9).

Все возможные расширения также перечислены в синтакс-помощнике. У каждого из них свой состав параметров, которые они добавляют к параметрам формы (рис. 3.10).

Какое именно расширение будет добавлено к конкретной форме, зависит от типа *основного реквизита формы*. Например, если форма отображает данные элемента справочника, то основной реквизит такой формы будет иметь тип `СправочникОбъект.<имя>`, а к форме будет добавлено *расширение справочника*.

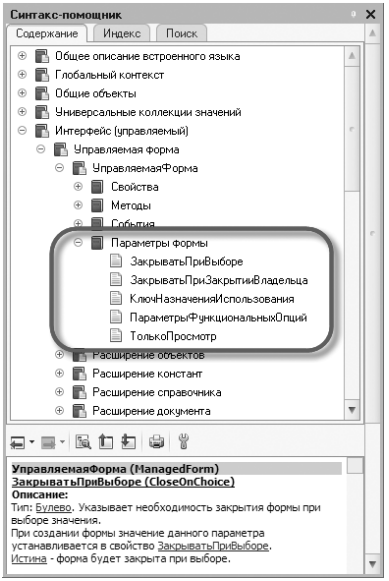


Рис. 3.9. Параметры формы в синтакс-помощнике

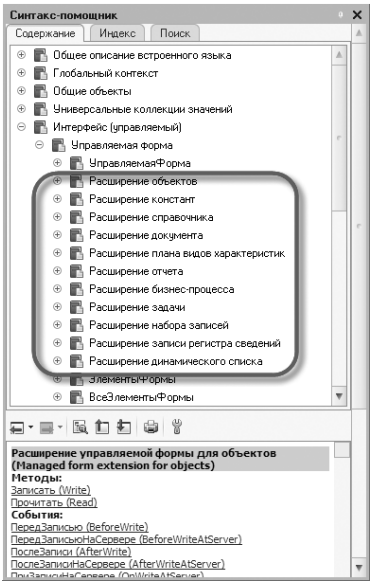


Рис. 3.10. Расширения формы в синтакс-помощнике

У расширения справочника есть параметр ЭтоГруппа. Если в этот параметр передать значение Истина, то будет создана именно новая группа, а не новый элемент.

Параметры формы можно передать вторым параметром метода ОткрытьФорму(). Так как может быть указано сразу несколько параметров, то они передаются в виде структуры. Каждый элемент этой структуры содержит имя параметра и его значение (листинг 3.4).

Пример можно посмотреть в демонстрационной базе «Открытие форм», панель функций текущего раздела, группа команд Открыть основную форму, глобальная команда Группы.

Листинг 3.4. Открытие основной формы новой группы

```
ПараметрыФормы = Новый Структура("ЭтоГруппа", Истина);
ОткрытьФорму("Справочник.Товары.ФормаГруппы", ПараметрыФормы);
```

Произвольная форма

Открыть форму, не назначенную основной для справочника Товары, можно следующим образом (листинг 3.5):

Листинг 3.5. Открытие произвольной формы

```
ОткрытьФорму("Справочник.Товары.Форма.ПроизвольнаяФормаТовара");
```

Здесь первым параметром метода также передается полное имя той формы, которую нужно открыть. Но поскольку открываемая форма не назначена основной, то после полного имени объекта конфигурации (Справочник.Товары) нужно указать слово Форма, а затем – имя нужной формы (ПроизвольнаяФормаТовара), как она названа в конфигураторе.

Пример можно посмотреть в демонстрационной базе «Открытие форм», панель функций текущего раздела, группа команд Сервис, глобальная команда Открыть произвольную форму.

Форма существующего объекта

Чтобы открыть форму существующего элемента, нужно указать, какой именно элемент нас интересует. Для этого также используются параметры формы.

У расширения справочника есть параметр Ключ. Если в этот параметр передать ссылку на существующий элемент справочника, то будет открыта форма именно этого элемента, а не нового.

Например, если из формы списка организаций нужно открыть форму той организации, на которой находится курсор, сделать это можно следующим способом (листинг 3.6):

Листинг 3.6. Открытие основной формы существующего объекта

```
СсылкаНаЭлементСправочника = Элементы.Список.ТекущаяСтрока;
ПараметрыФормы = Новый Структура("Ключ", СсылкаНаЭлементСправочника);
ОткрытьФорму("Справочник.Организации.ФормаОбъекта", ПараметрыФормы);
```

Чтобы получить ссылку на выделенный элемент списка, мы обращаемся к элементу формы Список и получаем его свойство ТекущаяСтрока.

Затем мы создаем структуру ПараметрыФормы для передачи параметров в открываемую форму. В этой структуре будет единственный элемент с ключом Ключ и значением полученной ссылки.

Пример можно посмотреть в демонстрационной базе «Открытие форм», список организаций, команда Открыть форму элемента. Это локальная команда формы списка справочника Организации.

После этого открывается форма объекта в соответствии с переданной структурой параметров.

В заключение следует заметить, что в том случае, когда кроме ссылки на объект никакие параметры в открываемую форму передавать не нужно и нужна именно основная форма существующего объекта, можно использовать более простой способ – с помощью функции глобального контекста ПоказатьЗначение() (листинг 3.7).

Листинг 3.7. Использование метода «ПоказатьЗначение()»

```
ПоказатьЗначение(, Элементы.Список.ТекущаяСтрока);
```

Такой способ обладает меньшей универсальностью, но в некоторых случаях он экономит много времени и сил. Например, в рассмотренном нами случае одна такая строка может заменить всю написанную нами ранее процедуру из трех строк.

Однако нужно внимательно относиться к использованию этого метода. В некоторых случаях он может вызывать дополнительные обращения к серверу, например когда нужно открыть форму элемента иерархического справочника. Платформа будет выполнять дополнительное обращение к серверу, для того чтобы определить, является элемент группой или нет, так как для элемента и для группы нужно открывать разные формы.

В то же время эта информация может быть заранее известна разработчику внутри его алгоритма, и, используя метод `ОткрытьФорму()`, разработчик может открывать нужную форму без дополнительных обращений к серверу.

Пример, подробно описывающий эту ситуацию, можно посмотреть в разделе «Использование стандартных полей запроса в динамических списках на клиенте» на стр. 803.

Открыть список, чтобы курсор был на нужном элементе

Одна из типичных задач при работе с формами списков – это открытие формы с одновременным позиционированием на конкретном элементе списка.

Из формы объекта это можно сделать интерактивно с помощью стандартной команды Показать в списке из подменю Еще. Но если это необходимо, можно спозиционироваться на конкретном элементе списка программно.

Для этого можно использовать параметр `ТекущаяСтрока`, который поставляется расширением динамического списка (рис. 3.11).

Например, для того чтобы сразу увидеть открытый элемент в списке товаров, нужно, находясь в форме элемента справочника Товары, открыть форму списка товаров так, чтобы курсор находился на этом элементе.

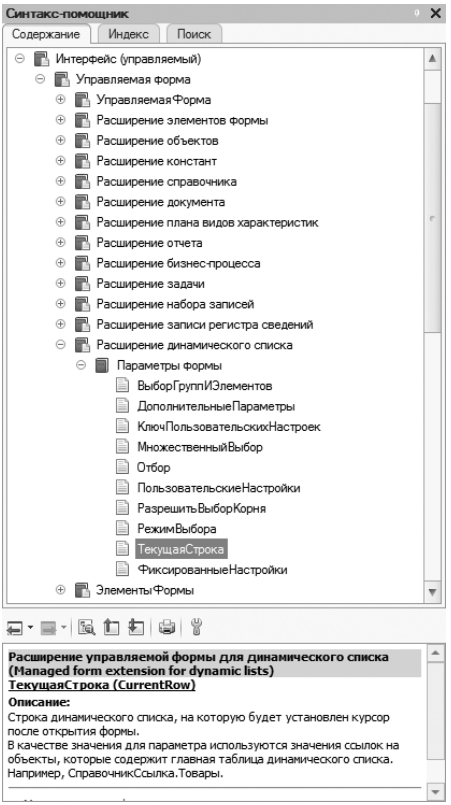


Рис. 3.11. Параметр «ТекущаяСтрока» в синтакс-помощнике

Для этого можно создать локальную команду формы элемента, которая будет выполнять следующий код (листинг 3.8).

Листинг 3.8. Открытие формы списка с позиционированием на некотором элементе

```
ПараметрыФормы = Новый Структура("ТекущаяСтрока", Объект.Ссылка);
ОткрытьФорму("Справочник.Товары.ФормаСписка", ПараметрыФормы);
```

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Открытие форм», форма элемента справочника Товары, команда Найти в списке. Это локальная команда формы элемента справочника Товары.

Список подчиненного справочника с отбором по владельцу

Другая типичная задача – открытие списка подчиненного справочника.

Стандартно команды для открытия подчиненных списков или других связанных данных можно выполнить из панели навигации окна, в котором открыта основная форма владельца. Но если необходимо (например, нужно открыть подчиненный список в отдельном окне), то можно реализовать эту задачу программно.

Для этого можно использовать параметр `Отбор`, который также поставляется расширением динамического списка.

В этот параметр нужно передать условие на отбор элементов подчиненного справочника по некоторому владельцу.

Например, это может выглядеть следующим образом. Справочник РасчетныеСчета подчинен справочнику Организации. В форме элемента справочника Организации (ссылка на этот элемент доступна как `Объект.Ссылка`) можно выполнить следующий код (листинг 3.9).

Листинг 3.9. Открытие формы подчиненного справочника с отбором по владельцу

```
УсловияОтбора = Новый Структура("Владелец", Объект.Ссылка)
ПараметрыФормы = Новый Структура("Отбор", УсловияОтбора);
ОткрытьФорму("Справочник.РасчетныеСчета.ФормаСписка", ПараметрыФормы);
```

Сначала создается структура, содержащая условия отбора (`УсловияОтбора`). В нашем случае условие будет одно, но, вообще говоря, их может быть несколько.

Затем эта структура устанавливается как значение параметра `Отбор` формы.

И в заключение открывается форма списка подчиненного справочника, в которую передается структура параметров формы.

Пример можно посмотреть в демонстрационной базе «Открытие форм», форма элемента справочника Организации, команда Показать расчетные счета. Это локальная команда формы элемента справочника Организации.

Передача параметров в произвольный запрос динамического списка

Еще одна задача, которая может возникать при открытии формы, заключается в том, что динамический список, содержащийся в форме, может содержать произвольный запрос с параметрами. И при открытии формы нужно передавать в этот список конкретные значения этих параметров.

Эта задача тоже решается с помощью параметров формы. Только это будут уже не стандартные параметры формы, предоставляемые платформой, а параметры формы, созданные разработчиком в конфигураторе.

В эти параметры нужно передать требуемые значения, а в обработчике события формы При создании на сервере установить эти значения параметрам запроса в динамическом списке.

Рассмотрим эту задачу на примере списка регистра ЦеныТоваров. В форме списка этого регистра содержится динамический список с произвольным запросом (листинг 3.10).

Листинг 3.10. Запрос динамического списка

```
ВЫБРАТЬ
    РегистрСведенийЦеныТоваров.Период,
    РегистрСведенийЦеныТоваров.Товар,
    РегистрСведенийЦеныТоваров.Цена
ИЗ
    РегистрСведений.ЦеныТоваров КАК РегистрСведенийЦеныТоваров
ГДЕ
    РегистрСведенийЦеныТоваров.Товар = &Товар
    И РегистрСведенийЦеныТоваров.Период >= &НачалоПериода
    И РегистрСведенийЦеныТоваров.Период <= &КонецПериода
```

У этого запроса есть три параметра: Товар, НачалоПериода и КонецПериода. Для нормального функционирования формы значения этих параметров должны быть заданы при открытии формы, иначе в результате открытия формы будет получена ошибка.

В форме элемента справочника Товары есть команда Цены в декабре, которая показывает цены на этот товар, установленные только в декабре 2009 года.

Пример можно посмотреть в демонстрационной базе «Открытие форм», форма элемента справочника Товары, команда Цены в декабре. Это локальная команда формы элемента справочника Товары, она открывает список регистра ЦеныТоваров.

Чтобы передать собственные параметры в форму, желательно создать их в редакторе формы. Вообще говоря, делать это не обязательно. Любые параметры, переданные в метод `ОткрытьФорму()`, будут доступны в обработчике события формы При создании на сервере. Но все же лучше создать их у формы в явном виде. Тогда, во-первых, в модуле формы не придется проверять, существуют такие параметры или нет. А во-вторых, явное создание параметров формы облегчает поддержку прикладного решения теми разработчиками, которые не участвовали в его создании.

Итак, создадим у формы списка регистра сведений три параметра: НачалоПериода, КонецПериода (тип `Дата`) и Товар (тип `СправочникСсылка.Товары`), рис. 3.12.

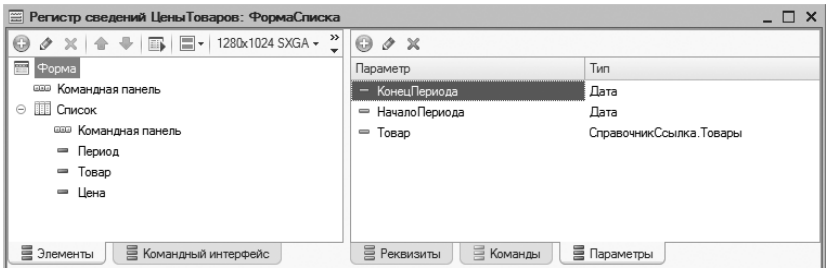


Рис. 3.12. Параметры формы

Теперь в форме элемента справочника Товары создадим команду ЦеныВДекабре со следующим текстом (листинг 3.11).

Листинг 3.11. Текст команды «ЦеныВДекабре»

```
ПараметрыФормы = Новый Структура(
                "НачалоПериода, КонецПериода, Товар", '20091201000000', '20091231235959', Объект.Ссылка);
ОткрытьФорму("РегистрСведений.ЦеныТоваров.Форма.ФормаСписка", ПараметрыФормы);
```

Этой командой мы открываем форму списка регистра сведений и передаем ей значения трех параметров.

В самой форме регистра сведений, в обработчике события формы При создании на сервере, установим переданные значения в качестве параметров произвольного запроса динамического списка следующим образом (листинг 3.12).

Листинг 3.12. Обработчик события «При создании на сервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Список.Параметры.УстановитьЗначениеПараметра("НачалоПериода", Параметры.НачалоПериода);
    Список.Параметры.УстановитьЗначениеПараметра("КонецПериода", Параметры.КонецПериода);
    Список.Параметры.УстановитьЗначениеПараметра("Товар", Параметры.Товар);

КонецПроцедуры
```

Метод «ПолучитьФорму()»

Кроме метода `ОткрытьФорму()` существует и другой метод глобального контекста, который может быть использован для открытия форм, – `ПолучитьФорму()`.

По составу параметров этот метод очень похож на `ОткрытьФорму()`. Разница заключается в том, что `ПолучитьФорму()` лишь возвращает форму, не открывая ее. В дальнейшем эта форма может быть открыта с помощью своего метода `Открыть()`.

Такой способ открытия формы можно использовать в тех случаях, когда должны выполняться различные сложные действия в зависимости от того, из какого фрагмента кода открывается форма. В этом случае после получения формы может быть, например, вызвана ее экспортируемая процедура, выполняющая настройку формы в зависимости от «контекста» ее вызова (листинг 3.13).

Листинг 3.13. Получение и открытие формы

```
Перем ССылкаНаТаблицуСтилей;

ФормаРедактора = ПолучитьФорму("Обработка.УниверсальныйРедактор.Форма");
ФормаРедактора.РедактироватьHTMLТекст("Пример HTML-текста", ССылкаНаТаблицуСтилей);
ФормаРедактора.Открыть();
```

В данном примере используется форма обработки УниверсальныйРедактор. Подразумевается, что обработка может редактировать как обычный текст, так и HTML-текст. Также подразумевается, что в зависимости от того, какой текст передается для редактирования, в самой обработке он определенным образом оформляется и подготавливается для редактирования.

Пример можно посмотреть в демонстрационной базе «Открытие форм», панель функций текущего раздела, группа команд Сервис, глобальная команда Открыть текст в редакторе.

Так как эта подготовка текста к редактированию не зависит от контекста, в котором вызван редактор, а определяется лишь видом редактируемого текста, в модуле формы этой обработки существуют две экспортируемые процедуры, позволяющие получить и подготовить к редактированию обычный текст (`РедактироватьОбычныйТекст()`) и HTML-текст (`РедактироватьHTMLТекст()`).

В приведенном примере как раз получается форма этой обработки, и с помощью экспортируемой процедуры `РедактироватьHTMLТекст()` ей передается текст, который нужно подготовить к редактированию. После этого форма обработки открывается.

Однако надо помнить, что использовать метод `ПолучитьФорму()` надо только в том случае, если это действительно необходимо. В остальных случаях рекомендуется использовать метод `ОткрытьФорму()`, передавая необходимые данные при открытии формы в качестве параметров. Такая рекомендация продиктована соображениями повышения устойчивости выполнения кода, а также сохранения единой стилистики кода прикладных решений.

Открытие формы в блокирующем режиме без использования модальности

Во многих случаях форма открывается для того, чтобы пользователь обязательно выполнил какие-то действия или ввел какие-то данные, без которых дальнейшая работа программы будет невозможна.

Раньше в таких случаях использовалось открытие формы в модальном режиме методом `ОткрытьФормуМодально()`. При этом полностью

блокировался весь остальной интерфейс программы, а исполнение программного кода останавливалось до тех пор, пока пользователь не закроет модальное окно.

Однако с развитием облачных технологий, с появлением веб-клиента и с переходом «1С:Предприятия» на мобильные платформы использование модальных окон стало вызывать большое количество неудобств и проблем.

Основные неудобства заключались в том, что для нормальной работы веб-клиента на обычных компьютерах требовалась предварительная настройка браузера. А на мобильных устройствах браузеры вообще не поддерживают работу с модальными окнами.

Поэтому в «1С:Предприятии» начиная с версии 8.3.3 реализован специальный режим работы интерфейса без использования модальных окон. Он определяется свойством конфигурации РежимИспользованияМодальности, которое стандартно установлено в значение Не использовать. Все новые конфигурации рекомендуется разрабатывать именно в этом режиме. А если вы хотите, чтобы конфигурация работала в веб-клиенте, в модели сервиса (например, 1cFresh) или на мобильных устройствах, то отказ от модальности – обязательное условие разработки.

В этом режиме работы интерфейса для пользователя все выглядит точно так же, как и раньше. Окно, которое раньше было бы модальным, рисуется в пределах родительского окна и точно так же блокирует весь остальной интерфейс. Таким образом обеспечивается «модальность» для пользователя. Он не сможет выполнить никакие другие действия, пока не закроет *блокирующее* окно.

Однако для разработчика в момент отображения блокирующего окна исполнение программного кода не останавливается. Программный код продолжает выполняться дальше. Это означает, что алгоритм, который обычно являлся одним целым, теперь придется разделить на две части: одну, которая заканчивается открытием блокирующего окна, и вторую, которая будет выполнена тогда, когда пользователь закроет это окно.

Чтобы система знала, с какого места продолжать исполнение программного кода, блокирующему окну передается имя процедуры, которая должна быть выполнена тогда, когда пользователь закроет это окно. Именно с этой процедуры и продолжится выполнение программного кода.

Для пояснения сказанного рассмотрим два примера.

Блокирующая форма может быть закрыта с помощью стандартных интерактивных команд формы либо из встроенного языка методом `Закрыть()`. В первом случае мы можем проанализировать *код возврата диалога*, соответствующий команде, с помощью которой блокирующая форма была закрыта.

Если форма закрывается из встроенного языка методом `Закрыть()`, то в параметр этого метода разработчик может передать произвольное значение, которое будет возвращено в вызывающий код.

Рассмотрим эти возможности на примере обработки, которая из своей основной формы открывает две другие формы для получения дополнительных данных.

Пример можно посмотреть в демонстрационной базе «Открытие форм», обработка ЗаполнениеДанных.

Сначала рассмотрим, как возвращать результаты выполнения стандартных интерактивных команд формы.

Для этого кроме основной формы обработки (Форма) создадим форму ПараметрыЗаполнения. Эта форма будет открываться в блокирующем режиме из основной формы и возвращать коды стандартных команд.

Затем в основной форме обработки создадим команду ЗадатьПараметрыЗаполнения и расположим ее в подгруппе Дополнительно в командной панели формы.

Чтобы открыть форму в блокирующем режиме, вместо метода `ОткрытьФормуМодально()` нужно использовать метод `ОткрытьФорму()`. И передавать туда в качестве последних двух параметров объект `ОписаниеОповещения`, указывающий на процедуру модуля, которая будет выполнена после закрытия блокирующего окна, и режим открытия этого окна, определяющий, будет ли заблокирован весь остальной интерфейс приложения или только окно, из которого открыта блокирующая форма.

Если форма всегда будет открываться только в блокирующем режиме, в конфигураторе можно сразу установить ее свойство РежимОткрытияОкна в значение Блокировать весь интерфейс. Но если форму нужно открывать то в независимом, то в блокирующем режиме, то можно свойство РежимОткрытияОкна оставить в стандартном значении Независимый. А при открытии формы в явном виде указывать нужный режим открытия окна последним параметром в методе ОткрытьФорму().

В обработчике команды ЗадатьПараметрыЗаполнения напишем следующий код (листинг 3.14).

Листинг 3.14. Обработчик команды «ЗадатьПараметрыЗаполнения»

```
ОписаниеОповещения = Новый ОписаниеОповещения(
                        "ЗадатьПараметрыЗаполненияЗавершение", ЭтотОбъект);
ОткрытьФорму("Обработка.ЗаполнениеДанных.Форма.ПараметрыЗаполнения",,,,,,
                        ОписаниеОповещения, РежимОткрытияОкнаФормы.БлокироватьВесьИнтерфейс);
```

В этом обработчике сначала создается объект ОписаниеОповещения, в конструкторе которого первым параметром указывается имя процедуры обработки оповещения, которая будет выполнена после закрытия блокирующего окна. А во втором параметре указывается, в каком модуле расположена эта процедура.

Процедура обработки оповещения может располагаться в модуле формы, в общем неглобальном клиентском модуле или модуле команды. В данном случае эта процедура располагается в модуле нашей формы, на что указывает ссылка ЭтотОбъект.

После этого методом ОткрытьФорму() форма ПараметрыЗаполнения открывается в блокирующем режиме. В этот метод передается объект ОписаниеОповещения, а также режим открытия окна, установленный в значение Блокировать весь интерфейс.

При этом весь остальной интерфейс приложения будет заблокирован, пока пользователь не закроет блокирующее окно. Таким образом мы полностью моделируем для пользователя старое, модальное поведение.

Но в плане поддержки однотипного поведения с другими веб-приложениями лучше использовать режим открытия окна в значении

Блокировать окно владельца. Тогда окно владельца будет заблокировано, но остальной интерфейс прикладного решения для пользователя будет доступен.

Такой режим открытия окна в зависимости от ситуации может иметь и свои плюсы, и свои минусы. Однако надо понимать, что открытие блокирующего окна в режиме блокирования окна владельца больше подходит к формату веб-приложений.

Теперь в форме ПараметрыЗаполнения в командную панель формы перенесем три стандартные команды: OK, Пропустить, Отмена (рис. 3.13).

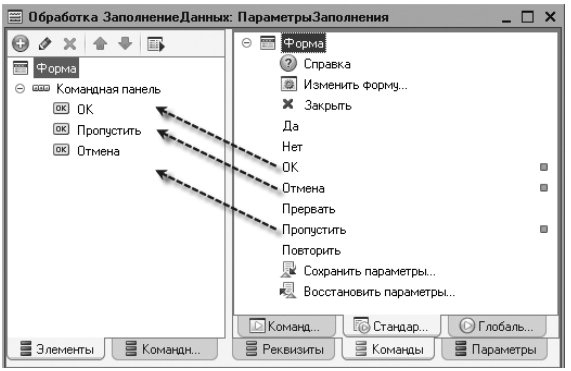


Рис. 3.13. Кнопки стандартных команд в командной панели

После закрытия формы ПараметрыЗаполнения одной из этих стандартных команд будет выполнена процедура обработки оповещения `ЗадатьПараметрыЗаполненияЗавершение()`. В параметре `Результат` этой процедуры будет содержаться код возврата диалога, значения которого описаны в системном перечислении `КодВозвратаДиалога`.

Расположим процедуру обработки оповещения в модуле формы исходной обработки и проанализируем возвращаемый результат, выведя в форму обработки сообщение, соответствующее нажатой кнопке в форме ПараметрыЗаполнения (листинг 3.15).

Листинг 3.15. Обработчик оповещения «ЗадатьПараметрыЗаполненияЗавершение»

```
&НаКлиенте
Процедура ЗадатьПараметрыЗаполненияЗавершение(Результат, ДополнительныеПараметры) Экспорт

	Сообщение = Новый СообщениеПользователю;

	Если Результат = КодВозвратаДиалога.ОК Тогда
		Сообщение.Текст = "В форме нажата стандартная кнопка ОК";

	ИначеЕсли Результат = КодВозвратаДиалога.Пропустить Тогда
		Сообщение.Текст = "В форме нажата стандартная кнопка Пропустить";

	ИначеЕсли Результат = КодВозвратаДиалога.Отмена Тогда
		Сообщение.Текст = "В форме нажата стандартная кнопка Отмена";

	Иначе
		Сообщение.Текст = "Обработка этой команды в форме не предусмотрена";
	КонецЕсли;

	Сообщение.Сообщить();

КонецПроцедуры
```

Обратите внимание, что процедура обработки оповещения должна быть объявлена как экспортная, с использованием ключевого слова Экспорт.

Теперь рассмотрим вторую возможность – возврат произвольного значения из блокирующей формы.

Для этого создадим еще одну форму обработки – ДобавлениеТекста. В ней создадим строковый реквизит (Текст) и поле для его редактирования. Добавим команду ЗакончитьВводТекста, по которой будем возвращать введенный пользователем текст (рис. 3.14).

Обработчик команды ЗакончитьВводТекста будет выглядеть совсем просто (листинг 3.16).

Листинг 3.16. Обработчик команды «ЗакончитьВводТекста»

```
Закрыть(Текст);
```

А в основной форме обработки добавим команду ДобавитьТекст, которая будет открывать форму ДобавлениеТекста в блокирующем режиме (листинг 3.17).

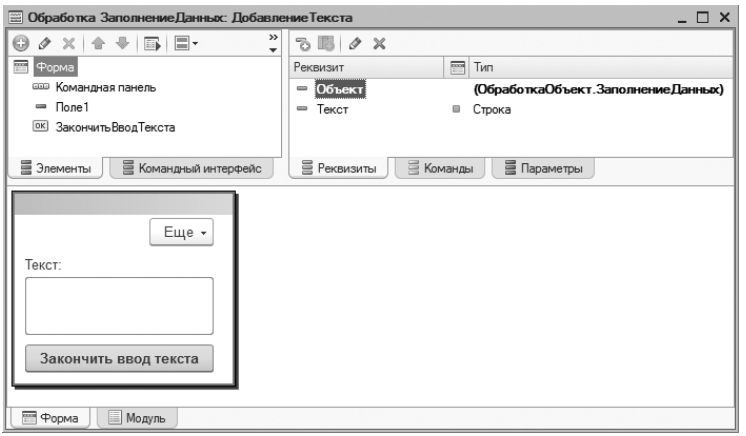


Рис. 3.14. Команда «ЗакончитьВводТекста» в форме «ДобавлениеТекста»

Листинг 3.17. Обработчик команды «ДобавитьТекст»

```
ОписаниеОповещения = Новый ОписаниеОповещения("ДобавитьТекстЗавершение", ЭтотОбъект);
ОткрытьФорму("Обработка.ЗаполнениеДанных.Форма.ДобавлениеТекста",,,,,,
                    ОписаниеОповещения, РежимОткрытияОкнаФормы.БлокироватьОкноВладельца);
```

Обратите внимание, что в данном случае блокирующее окно открыто в режиме блокирования окна владельца.

После закрытия формы `ДобавлениеТекста` будет выполнена процедура обработки оповещения `ДобавитьТекстЗавершение()`. В параметре `РезультатЗакрытия` этой процедуры будет содержаться текст, введенный пользователем в блокирующей форме.

Расположим процедуру обработки оповещения в модуле формы исходной обработки и выведем этот текст в сообщение (листинг 3.18).

Листинг 3.18. Обработчик команды «ДобавитьТекстЗавершение»

```
&НаКлиенте
Процедура ДобавитьТекстЗавершение(РезультатЗакрытия, ДополнительныеПараметры) Экспорт

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "В форме введен текст: " + РезультатЗакрытия;
    Сообщение.Сообщить();

КонецПроцедуры
```

К примерам реализации отказа от модальности мы еще вернемся (ПоказатьПредупреждение() – листинг 3.61, НачатьПомещениеФайла() – листинг 3.129, ПоказатьВопрос() – листинг 3.34, ПоказатьВопрос() в обработчике – листинг 3.164.).

Открытие и запуск отчета

Последняя задача, которую мы рассмотрим в этой главе, это открытие формы отчета и выполнение этого отчета без дополнительных действий со стороны пользователя.

Например, есть отчет ЦеныТоваров. Он показывает последние цены, установленные на товары.

Задача в том, чтобы по команде в форме товаров сразу запускать этот отчет с отбором, установленным по тому товару, на котором находится курсор в списке.

Выполнить эту задачу помогут два параметра, которые поставляются расширением отчета: Отбор и СформироватьПриОткрытии. Параметр Отбор позволяет установить отбор в отчете, задав значения полей или параметров отчета. А параметр СформироватьПриОткрытии позволяет сразу же выполнить отчет, после того как его форма будет получена на сервере.

Итак, создадим форму списка справочника Товары и добавим в нее команду ЦенаТовара. Текст обработчика этой команды будет выглядеть следующим образом (листинг 3.19).

Пример можно посмотреть в демонстрационной базе «Открытие форм», форма списка справочника Товары, команда ЦенаТовара.
Это локальная команда формы списка справочника Товары.

Листинг 3.19. Обработчик команды «ЦенаТовара»

```
УсловияОтбора = Новый Структура("Товар" Элементы.Список.ТекущаяСтрока);
ПараметрыФормы = Новый Структура("Отбор, СформироватьПриОткрытии", УсловияОтбора, Истина);
ОткрытьФорму("Отчет.ЦеныТоваров.ФормаОбъекта", ПараметрыФормы);
```

Сначала создаем структуру УсловияОтбора, которая задаст единственное условие отбора – по тому товару, на котором установлен курсор в списке (Элементы.Список.ТекущаяСтрока).

Затем сформируем структуру для параметров формы (ПараметрыФормы), описав в ней значения двух параметров: Отбор и СформироватьПриОткрытии.

В заключение, как и раньше, открываем форму отчета (ОткрытьФорму()), указав имя формы и параметры, предназначенные для открываемой формы.

Переопределение открываемой формы

Иногда бывает нужно открывать различные формы в зависимости от выполнения каких-либо условий. Например, для элемента справочника Товары нужно открывать разные формы для собственно товара и для услуги.

Для решения этой задачи в модуле менеджера объекта следует создать обработчик события ОбработкаПолученияФормы. В этом обработчике нужно выполнить все необходимые проверки и принять решение о том, какую форму следует открывать. И соответствующим образом переопределить параметр обработчика ВыбраннаяФорма (листинг 3.20).

Пример можно посмотреть в демонстрационной базе «Открытие форм», в модуле менеджера справочника Товары.

Листинг 3.20. Обработчик «ОбработкаПолученияФормы» в модуле менеджера

```
Процедура ОбработкаПолученияФормы(ВидФормы, Параметры, ВыбраннаяФорма,
                                  ДополнительнаяИнформация, СтандартнаяОбработка)

    Если Параметры.Свойство("Ключ") Тогда
        Если ВидФормы = "ФормаОбъекта"
                И Параметры.Ключ.Родитель.Наименование = "Услуги" Тогда
            ВыбраннаяФорма = "Справочник.Товары.Форма.ПроизвольнаяФормаТовара";
            СтандартнаяОбработка = Ложь;
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике (для существующих товаров) сначала проверяется, что открывается форма объекта (параметр ВидФормы = "ФормаОбъекта"). Ссылка на объект передается в структуре параметров открываемой формы (Параметры.Ключ).

Поскольку у товара нет специального реквизита, определяющего его вид, будем считать, что все услуги собраны в отдельной папке справочника с наименованием «Услуги».

Если элемент справочника находится в этой папке, то мы переопределяем параметр `ВыбраннаяФорма`. Тем самым указываем форму, которая должна быть открыта вместо формы объекта – `ПроизвольнаяФормаТовара`. И параметр `СтандартнаяОбработка` устанавливаем в значение `Ложь`.

В результате для услуг будет открыта произвольная форма товара, основная форма товара будет открываться только для товаров, не содержащихся в папке «Услуги». Ну и, естественно, для вновь создаваемых товаров (у которых свойство `Ключ` в структуре параметров формы не определено) мы никаких проверок не делаем.

Если требуется открывать не форму услуги, а какую-то связанную с ней информацию (например, счет), то нужно также переопределить и параметр `Ключ`, чтобы он содержал ссылку на этот связанный объект.

Глава 3.4. Преобразование прикладных данных в данные формы

Встроенный язык содержит целый ряд объектов, предназначенных для чтения и модификации объектных данных, хранящихся в базе данных. Это такие типы, как `СправочникОбъект.<имя>`, `ДокументОбъект.<имя>` и т. д. Однако работа с этими типами данных возможна только в контексте сервера. Форма же и на клиенте, и на сервере для отображения таких объектных данных использует специальные универсальные типы.

Поэтому при открытии форм объектов платформа выполняет автоматическое преобразование данных из типов объектов в типы, поддерживаемые формой. При записи данных объектов из формы происходит обратное преобразование. Рассмотрим эти процессы подробнее.

Когда открывается форма существующего объекта, платформа читает его данные из базы данных и создает в памяти соответствующий этим данным объект встроенного языка (рис. 3.15). Например, `СправочникОбъект.<имя>`.

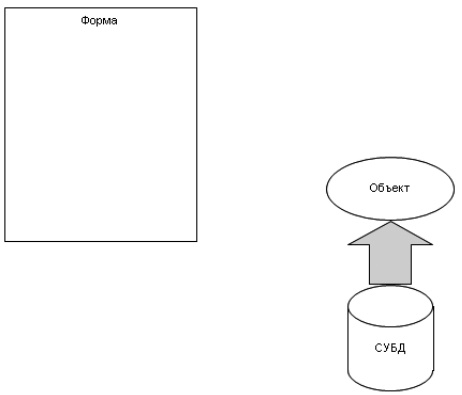


Рис. 3.15. Чтение данных существующего объекта

После этого данные объекта преобразуются в данные основного реквизита формы. Как правило, основной реквизит формы имеет имя Объект (рис. 3.16).

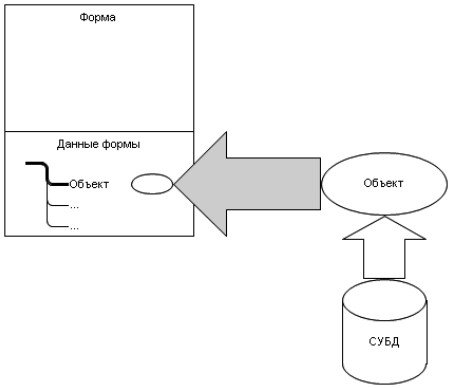


Рис. 3.16. Преобразование данных объекта в данные формы

После того как форма полностью подготовлена, она передается с сервера на клиент (рис. 3.17).

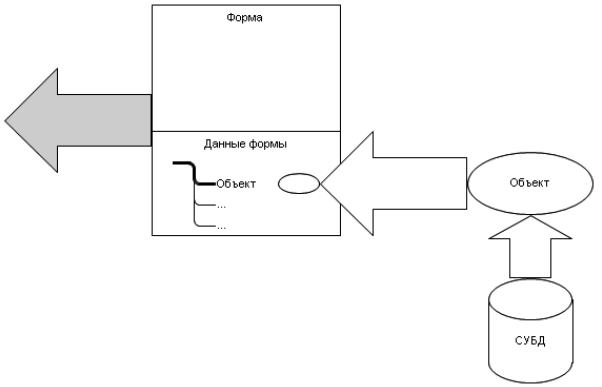


Рис. 3.17. Передача формы на клиент

Обратное преобразование выполняется аналогичным образом. Сначала форма передается с клиента на сервер (рис. 3.18).

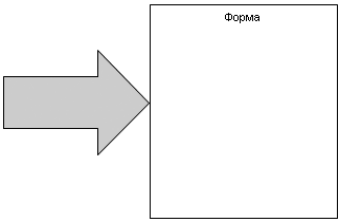


Рис. 3.18. Передача формы на сервер

После этого основной реквизит формы преобразуется в соответствующий объект встроенного языка, позволяющий модифицировать данные в базе данных. Например, `СправочникОбъект.<имя>` (рис. 3.19).

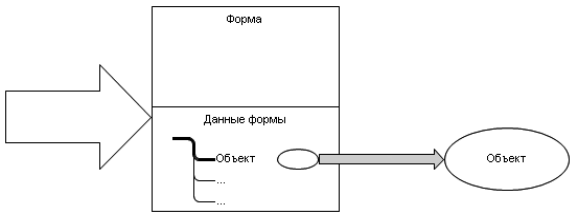


Рис. 3.19. Преобразование данных формы в прикладной объект

Затем данные объекта записываются в базу данных (рис. 3.20).

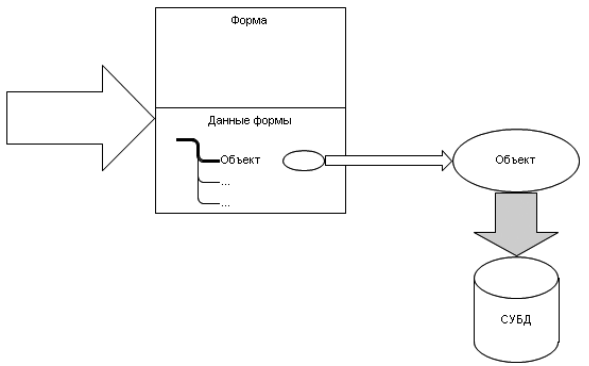


Рис. 3.20. Запись данных прикладного объекта в базу данных

Существуют четыре типа универсальных данных формы, в которые преобразуются типы, доступные только на сервере:

- ДанныеФормыСтруктура;
- ДанныеФормыКоллекция;
- ДанныеФормыДерево;
- ДанныеФормыСтруктураСКоллекцией.

Это «простые» типы данных – в том смысле, что они по сути представляют собой определенным образом организованные структуры данных и не содержат никакого специфического поведения, зависящего от данных.

В какой именно тип будут преобразованы серверные данные – зависит от самих этих данных. Преобразование выполняется в те универсальные типы, которые наиболее хорошо подходят для описания исходного типа данных.

Например, данные типа СправочникОбъект.Товары будут преобразованы в тип ДанныеФормыСтруктура (рис. 3.21).

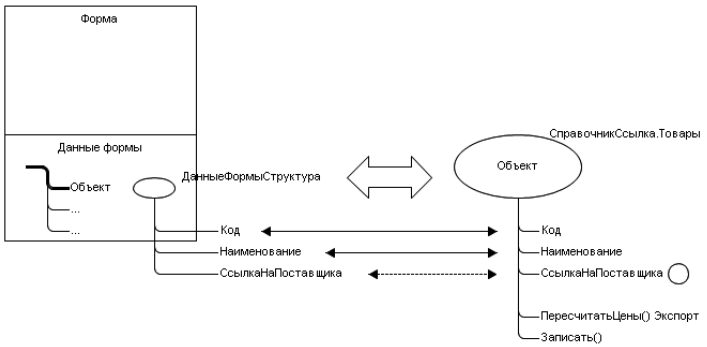


Рис. 3.21. Преобразование данных элемента справочника в «ДанныеФормыСтруктура»

Элементы этой структуры будут содержать значения реквизитов справочника Товары. Реквизиты примитивных типов в форме будут иметь тот же тип, что и у серверного объекта. Например, Код и Наименование будут иметь тип Строка.

Никаких специфических методов ДанныеФормыСтруктура иметь не будет. Это касается и стандартных методов объекта справочника (например, Записать()), и методов, описанных разработчиком в модуле объекта (например, экспортируемая процедура ПересчитатьЦены()).

Реквизиты ссылочного типа, например СсылкаНаПоставщика типа СправочникСсылка.Поставщики, будут иметь такой же тип, как и на сервере – СправочникСсылка.<имя>. Однако программный объект ссылки в форме в контексте клиента очень сильно ограничен в своих возможностях. Практически единственное, что можно сделать со ссылкой на клиенте, – это получить ее значение. Ну, и еще проверить, является ли эта ссылка пустой. Создание объектов от ссылки, получение значений через точку от ссылки на клиенте недоступны. Это возможно только на сервере.

Как уже говорилось выше, для системных процессов, таких как запись, проведение объекта из формы, подобное преобразование выполняется платформой автоматически. Однако встроенный язык при необходимости позволяет разработчику выполнять такие преобразования самостоятельно. Для этого используются процедуры глобального контекста ДанныеФормыВЗначение() и ЗначениеВДанныеФормы().

Аналогичные методы существуют и у формы: РеквизитФормыВЗначение() и ЗначениеВРеквизитФормы(). Подробнее они будут рассмотрены в главе 3.7 на стр. 486 и в главе 4.3 в разделе «Запись данных объекта в единой транзакции за один серверный вызов» на стр. 820 и в разделе «Реализация пересчета данных объекта в модуле объекта или в модуле формы в зависимости от логики объекта» на стр. 848.

Глава 3.5. Исполнение модуля формы на клиенте и на сервере

Как уже говорилось ранее, форма представляет собой особенный, специфический объект встроенного языка. Форма состоит из двух взаимодействующих между собой частей, каждая из которых выполняется в разных контекстах. Одна часть – в контексте сервера, а другая – в контексте клиента.

По этой причине все процедуры и функции, создаваемые в модуле формы, должны иметь явное указание на то, в каком контексте они будут исполняться.

Для этого используются *директивы компиляции*, набор которых определен во встроенном языке. Одна из возможных директив компиляции обязательно должна быть указана перед объявлением каждой процедуры или функции, находящейся в модуле формы.

Например, директива компиляции и процедура могут выглядеть следующим образом (рис. 3.22).

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    // Установка значения реквизита АдресКартинки.
    ФайлКартинки = Объект.ФайлКартинки;
    Если НЕ ФайлКартинки.Пустая() Тогда
        АдресКартинки = ПолучитьНавигационнуюСсылку(ФайлКартинки, "ДанныеФайла")
    КонецЕсли;

    ЗаполнитьХарактеристики();

    ОпределитьДоступность(ЭтаФорма);
КонецПроцедуры
```

Рис. 3.22. Директива компиляции в тексте модуля

На этом рисунке показана директива компиляции &НаСервере и процедура, обрабатывающая событие формы При создании на сервере.

Поскольку стандартные события вызываются платформой в заранее известных контекстах (на клиенте или на сервере), то и директивы компиляции, указываемые перед обработчиками этих событий, должны соответствовать контексту события.

Что же касается контекстов исполнения собственных процедур или функций – их определяет сам разработчик в зависимости от своих нужд.

Вообще в модуле формы могут быть использованы четыре различные директивы компиляции:

- `&НаКлиенте`;
- `&НаСервере`;
- `&НаСервереБезКонтекста`;
- `&НаКлиентеНаСервереБезКонтекста`.

Директива `&НаКлиенте` указывает, что процедура или функция будет исполняться в контексте клиентского приложения. В этой процедуре будет доступен весь контекст формы: реквизиты, элементы и параметры формы. Эта директива используется для всех обработчиков клиентских событий формы, а также для процедур, описывающих локальные команды формы. В дальнейшем процедуры, которым предшествует эта директива, мы будем называть иногда *клиентскими процедурами формы*.

Директива `&НаСервере` аналогична предыдущей – с той разницей, что исполнение кода происходит в контексте сервера. Эта директива используется для всех обработчиков серверных событий формы. Также эту директиву часто используют разработчики для собственных процедур, чтобы передать исполнение кода на сервер. В дальнейшем процедуры, которым предшествует эта директива, мы будем называть иногда *серверными процедурами формы*. А вызов такой процедуры из клиентской процедуры мы будем называть *контекстным серверным вызовом*.

Директива `&НаСервереБезКонтекста` также определяет, что код будет исполняться в контексте сервера, но при этом *контекст формы* (реквизиты, элементы, параметры) будет недоступен. Эту директиву также используют разработчики для собственных процедур, чтобы передать исполнение кода на сервер. Как и в предыдущем случае, такие процедуры мы будем называть иногда *серверными процедурами формы*. А вот вызов такой процедуры из клиентской процедуры мы будем называть *внеконтекстным серверным вызовом*.

И последняя директива, &НаКлиентеНаСервереБезКонтекста, определяет, что процедура или функция может исполняться как в контексте клиента, так и в контексте сервера. Эта директива используется редко. Обычно она требуется в тех случаях, когда нужно выполнять одинаковые действия как при создании формы, так и в процессе ее жизнедеятельности на клиенте. Тогда вместо двух одинаковых процедур, одна из которых исполняется на сервере, а другая – на клиенте, создается одна процедура с директивой &НаКлиентеНаСервереБезКонтекста.

Если по ошибке никакая директива компиляции не указана, то платформа считает, что используется директива &НаСервере.

Процесс исполнения кода на клиенте и сервере в формах соответствует общему подходу, принятому в «1С:Предприятии». Из клиентских процедур можно вызвать серверные, после их выполнения исполнение кода возвращается на клиент. Принудительно передать исполнение кода в обратную сторону, с сервера на клиент, нельзя, т. е. из серверных процедур нельзя вызвать клиентские.

Переменные модуля формы

В модуле формы директивы компиляции должны быть указаны не только перед определением процедур и функций, но и перед определением переменных.

Если перед описанием переменной используется директива компиляции &НаКлиенте, то такая переменная будет существовать с момента создания формы до момента ее закрытия. Причем только в клиентской части формы. Из серверных процедур формы она доступна не будет.

Если перед описанием переменной используется директива компиляции &НаСервере, то такая переменная будет существовать только на период вызова и исполнения какой-либо серверной процедуры или функции. После того как исполнение кода вернется на клиент, модуль, исполнявшийся на сервере, будет уничтожен вместе с этой переменной (рис. 3.23).

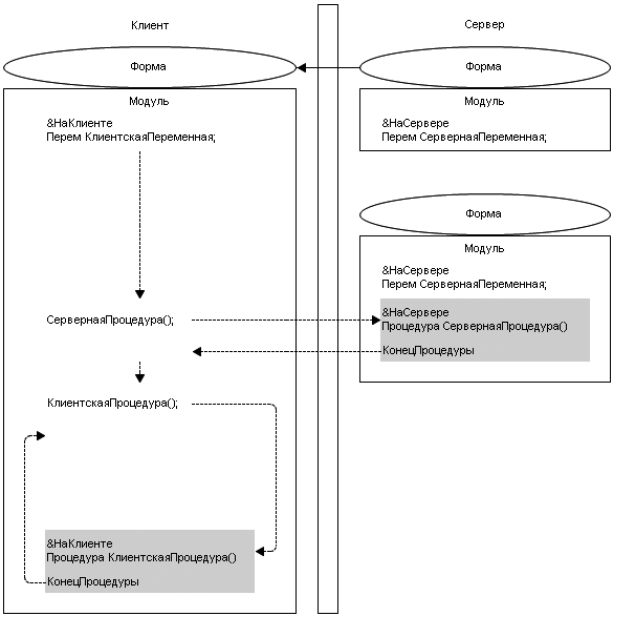


Рис. 3.23. Однократная компиляция клиентской части модуля и многократная компиляция серверной части модуля

Если эти замечания трудно воспринимаются прямо сейчас, можно прочитать несколько следующих разделов и вернуться еще раз к этому фрагменту. Он станет понятнее.

Почему важно понимать эту особенность? Потому что порой переменные модуля объявляются в явном виде, без указания директивы компиляции. В этом случае, как уже говорилось выше, платформа будет использовать директиву `&НаСервере`.

Можно пытаться в одной серверной процедуре устанавливать значение этой переменной и читать ее в другой. Но так получится только в том случае, если все эти процедуры выполняются в одном серверном вызове. Если серверные вызовы разные, переменная будет инициализироваться каждый раз заново.

В то же время на клиенте можно использовать переменные модуля для хранения значений на все время жизни формы, но для этого

при объявлении таких переменных обязательно должна быть указана директива компиляции &НаКлиенте.

Передавать значения с клиента на сервер с помощью переменных модуля формы нельзя. Хранить значения на сервере в переменных модуля также не получится. Для этого нужно либо использовать реквизиты формы (и выполнять контекстные серверные вызовы), либо передавать значения в параметрах вызываемой процедуры/функции.

Что касается процесса записи объекта из формы, тут есть одна удобная возможность, позволяющая использовать некоторый произвольный набор значений (структуру), доступный одновременно как в клиентских, так и в серверных процедурах. Речь идет о *параметрах записи* – это структура, указываемая в методе Записать() расширения формы. Подробнее об этом можно прочитать в разделе «Параметры записи» на стр. 510.

Экспортируемые процедуры формы

Разработчик может создать в модуле формы экспортируемые процедуры или функции. Они будут доступны как методы программного объекта Форма во встроенном языке.

Для таких процедур/функций также нужно указывать директивы компиляции. При этом практический смысл есть, наверное, в использовании лишь единственной директивы – &НаКлиенте.

Поясним почему. Как мы уже знаем, объект формы создается методами ОткрытьФорму() или ПолучитьФорму(). Оба они доступны только в контексте клиента. Поэтому созданная форма также будет существовать в контексте клиента. Соответственно, в этом же контексте должны быть определены ее экспортируемые процедуры/функции.

Можно вспомнить пример, который мы рассматривали при изучении открытия форм в разделе «Метод "ПолучитьФорму()"» на стр. 460.

В нем глобальная команда получала форму обработки, вызывала ее экспортируемую процедуру и потом открывала форму. Глобальная команда выполняется в контексте клиента, полученная форма существует в контексте клиента, поэтому и экспортируемые процедуры

этой формы (см. обработка УниверсальныйРедактор, форма ФормаРедактора) описаны с директивами компиляции &НаКлиенте.

В принципе программный объект формы может существовать как на клиенте, так и на сервере. Однако для чего, в какой ситуации может потребоваться обращаться к нему «извне» на сервере – сложно представить.

Поэтому основное замечание этого раздела заключается в том, что при написании собственных экспортируемых процедур/функций формы нужно не забывать указывать для них директиву компиляции &НаКлиенте. Если никакая директива указана не будет, платформа будет считать, что используется директива &НаСервере, и, значит, вызвать процедуру/функцию на клиенте вы не сможете.

Однако без особой необходимости лучше не располагать экспортируемые процедуры и функции в модулях форм. Для этого рекомендуется использовать модули объектов, модули менеджеров объектов и общие модули. Исключения составляют экспортируемые процедуры обработки оповещений.

Глава 3.6. Контекстные и внеконтекстные серверные вызовы

Теперь рассмотрим подробнее, что происходит при контекстных и при внеконтекстных серверных вызовах в форме. Для упрощения рассказа мы опустим тонкости оптимизации и кеширования, которые выполняет при этом платформа, так как на суть описываемых действий это не влияет.

Когда из клиентской процедуры/функции вызывается серверная (с директивой компиляции `&НаСервере`) процедура/функция, происходит передача всей формы на сервер.

Сначала контекст формы (реквизиты формы, элементы формы, параметры формы) специальным образом упаковываются и подготавливаются к передаче на сервер (рис. 3.24).

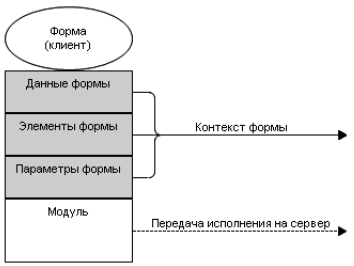


Рис. 3.24. Подготовка контекста формы и передача его на сервер

На рисунке темным цветом выделены те данные, которые, возможно, были изменены на клиенте.

На сервере создается серверная часть формы, полученный контекст разворачивается в данные, элементы и реквизиты формы, после чего инициализируется модуль формы (рис. 3.25).

После того как программный объект формы будет полностью подготовлен к работе, выполняется код той процедуры/функции, которая была вызвана на клиенте (рис. 3.26).

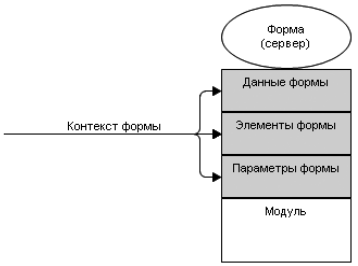


Рис. 3.25. Получение контекста формы и инициализация серверной части модуля формы

Рис. 3.26. Выполнение кода на встроенном языке из модуля формы

На рисунке темным цветом выделены те данные, которые, возможно, были изменены на сервере в процессе выполнения серверной процедуры.

После того как исполнение кода на сервере будет закончено, происходит обратное действие. Контекст формы упаковывается и передается обратно на клиент, в клиентскую часть формы (рис. 3.27).

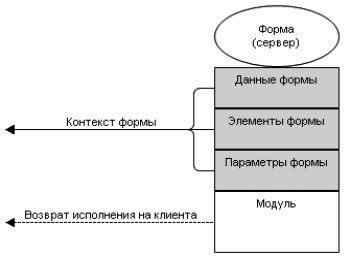


Рис. 3.27. Упаковка контекста формы и передача его на клиент

На клиенте происходит синхронизация контекста формы: имеющийся контекст заменяется контекстом, полученным с сервера. А на сервере происходит уничтожение серверной части формы (рис. 3.28).

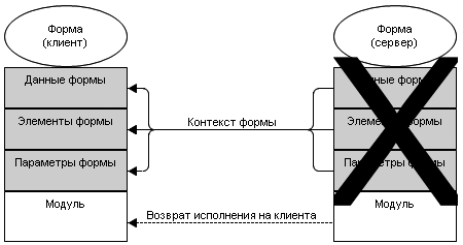


Рис. 3.28. Замена контекста на клиенте и уничтожение серверной части формы

Таким образом, контекстный вызов формы – довольно затратный и непростой процесс. Использовать все время только контекстные вызовы сервера было бы расточительно и непроизводительно. Поэтому контекстные вызовы сервера рекомендуется использовать тогда, когда обойтись без них очень трудно или невозможно вообще.

В то же время есть масса ситуаций, когда нужно выполнять некоторые действия на сервере, но контекст формы при этом не нужен. Для этого используются внеконтекстные серверные вызовы, когда из клиентской процедуры вызывается серверная процедура с директивой компиляции `&НаСервереБезКонтекста`.

В этом случае из модуля клиентской части формы выполнение кода сразу же передается на сервер, во внеконтекстную процедуру. После ее выполнения осуществляется возврат на клиент (рис. 3.29).

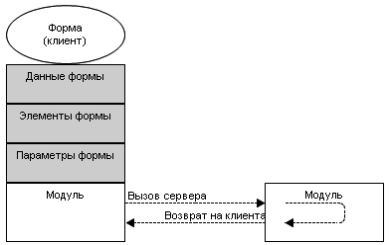


Рис. 3.29. Внеконтекстный вызов серверной процедуры/функции

Внеконтекстный вызов проще, эффективнее и производительнее. Именно его рекомендуется использовать в большинстве случаев.

Однако не стоит увлекаться фанатичной погоней за производительностью, осуществляя только внеконтекстные вызовы и передавая контекст формы в параметрах вызываемой процедуры.

Во-первых, всегда следует искать золотую середину между «читабельностью» программы и ее производительностью. Можно так «заоптимизировать» прикладное решение, что разобраться в нем через некоторое время не сможет уже никто, включая самого автора.

Во-вторых, контекстные вызовы сервера мы рассматривали упрощенно. На самом деле платформа очень серьезно оптимизирует объем данных, передаваемых между клиентом и сервером. Например, при открытии формы на клиент передается только часть данных, которую видит в данный момент пользователь. С клиента на сервер передаются не все данные, а только те, которые были изменены на клиенте. При изменении данных на сервере на клиент также передаются не все данные, а только те, которые были изменены.

Поэтому если на сервере требуется обращаться к большому массиву данных, которые хранятся в данных формы, то выгоднее выполнить именно контекстный серверный вызов. Потому что в этом случае платформа эффективно выполнит передачу данных формы на сторону сервера, там сконструирует соответствующий контекст и обратно перешлет данные на клиент.

Глава 3.7. Работа с данными объекта в форме

В этом разделе мы рассмотрим несколько простых примеров работы с данными объекта в форме. Они помогут понять общие подходы к основным действиям.

Пример 1

В качестве первого примера рассмотрим ситуацию, когда мы находимся в форме документа и нам нужно получить какой-либо его реквизит. Например, номер документа или значение реквизита Поставщик.

> Пример можно посмотреть в демонстрационной базе «Работа с данными объекта в форме», форма документа Накладная, группа команд Примеры, команда Значение реквизита. Это локальная команда формы документа Накладная.

Данные объекта, которые отображаются в форме (в нашем случае данные документа), находятся всегда в основном реквизите формы. В подавляющем большинстве случаев этот реквизит имеет имя Объект. По крайней мере именно такое имя дает ему платформа, когда с помощью конструктора создает форму объекта. В нашем случае это именно так (рис. 3.30).

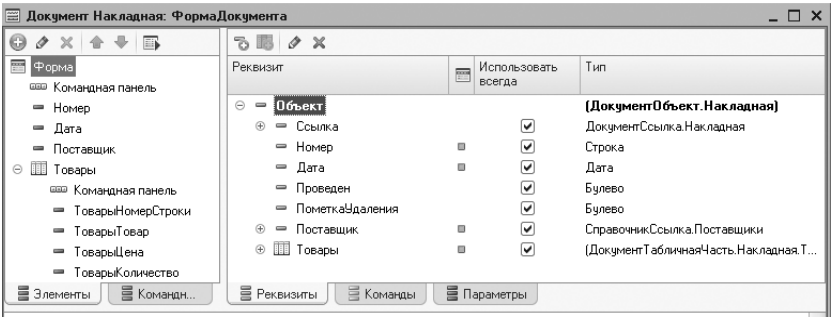


Рис. 3.30. Стандартное имя основного реквизита формы объекта

Основной реквизит формы содержит набор подчиненных реквизитов, соответствующих реквизитам прикладного объекта. Поэтому для того, чтобы обратиться, например, к номеру документа, нужно указать имя реквизита формы и через точку – имя подчиненного реквизита, хранящего номер документа.

В нашем случае, для того чтобы сообщить номер документа и поставщика, нужно будет выполнить следующий код (листинг 3.21).

Листинг 3.21. Получение значения реквизита формы

```
Сообщение = Новый СообщениеПользователю;
Сообщение.Текст = Объект.Номер + Символы.ПС + Строка(Объект.Поставщик);
Сообщение.Сообщить();
```

Номер документа имеет тип `Строка`, поэтому мы выводим его просто как `Объект.Номер`, а значение реквизита `Поставщик` имеет тип ссылки. Поэтому для того, чтобы включить его в текстовую строку, мы получаем строковое представление этого ссылочного значения, преобразуя его к типу `Строка`: `Строка(Объект.Поставщик)`.

Пример 2

Второй пример, который мы рассмотрим, будет заключаться в том, чтобы получить реквизит от ссылочного значения, хранящегося в реквизите формы. Как раз у документа Накладная есть реквизит ссылочного типа – Поставщик. Мы будем получать ИНН этого поставщика.

Пример можно посмотреть в демонстрационной базе «Работа с данными объекта в форме», форма документа Накладная, группа команд Примеры, команда Реквизит от ссылки. Это локальная команда формы документа Накладная.

Мы, как и в предыдущем примере, по-прежнему находимся в модуле формы на клиенте. В контексте клиента, как мы уже упоминали выше, ссылочные типы очень сильно ограничены в своих возможностях. В частности, нельзя получить значение реквизита от ссылки «через точку». Это можно сделать только в контексте сервера.

Поэтому для решения нашей задачи мы создадим серверную внеконтекстную функцию, которую вызовем с клиента и передадим

в нее ссылку на поставщика. В теле функции, на сервере, мы получим ИНН поставщика и вернем его на клиент.

Локальная команда формы, вызывающая серверную функцию, будет выглядеть следующим образом (листинг 3.22).

Листинг 3.22. Обработчик локальной команды формы

```
&НаКлиенте
Процедура РеквизитОтСсылки(Команда)

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = ПолучитьИННПоставщика(Объект.Поставщик);
    Сообщение.Сообщить();

КонецПроцедуры
```

Здесь в текст сообщения мы помещаем то, что вернет нам серверная функция `ПолучитьИННПоставщика()`. Эту функцию мы опишем в модуле формы следующим образом (листинг 3.23).

Листинг 3.23. Серверная функция для получения реквизита от ссылочного значения

```
&НаСервереБезКонтекста
Функция ПолучитьИННПоставщика(Поставщик)

    Возврат Поставщик.ИНН;

КонецФункции
```

То есть на сервере мы сразу же получаем значение реквизита «через точку» от полученной ссылки на поставщика.

Обратите внимание еще раз: эта функция не использует контекст формы. Потому что все, что нам нужно передать на сервер, – это только ссылка, и мы передаем ее в параметре функции. Использовать контекстный вызов и «гонять» на сервер весь контекст формы только ради того, чтобы на сервере взять из него значение одного реквизита формы, было бы слишком расточительно.

Пример 3

Третий пример, который мы рассмотрим, будет заключаться в вызове экспортируемой процедуры объекта. Наш документ как раз имеет одну такую процедуру, которая позволяет пересчитать все цены, содержащиеся в табличной части документа, и применить к ним скидку в 10 %.

Поскольку для вызова такой процедуры необходимо прежде всего иметь сам объект, мы поступим следующим образом.

Выполним контекстный серверный вызов. На сервере преобразуем основной реквизит формы в прикладной объект, выполним экспортируемую функцию этого объекта и преобразуем объект обратно в основной реквизит формы.

Пример можно посмотреть в демонстрационной базе «Работа с данными объекта в форме», форма документа Накладная, группа команд Примеры, команда Метод объекта. Это локальная команда формы документа Накладная.

В результате выполненные на сервере изменения данных формы будут автоматически переданы на клиент, когда на него вернется выполнение программного кода.

Локальная команда формы, вызывающая серверную процедуру, будет выглядеть следующим образом (листинг 3.24).

Листинг 3.24. Обработчик локальной команды формы

```
&НаКлиенте
Процедура МетодОбъекта(Команда)

    ПересчитатьЦеныНаСервере();

КонецПроцедуры
```

Контекстная серверная функция `ПересчитатьЦеныНаСервере()` будет выглядеть следующим образом (листинг 3.25).

Листинг 3.25. Серверная функция для вызова экспортируемой функции объекта

```
&НаСервере
Процедура ПересчитатьЦеныНаСервере()

    ОбъектДокумента = РеквизитФормыВЗначение("Объект", Тип("ДокументОбъект.Накладная"));
    ОбъектДокумента.НачислитьСкидку(10);
    ЗначениеВРеквизитФормы(ОбъектДокумента, "Объект");

КонецПроцедуры
```

В этой функции сначала мы преобразуем данные реквизита формы `Объект` в прикладной объект типа `ДокументОбъект.Накладная`.

После этого мы вызываем экспортируемую процедуру этого прикладного объекта – `НачислитьСкидку()`. В результате ее работы все цены, содержащиеся в табличной части объекта, будут уменьшены на 10 %.

Затем мы преобразуем данные прикладного объекта обратно в реквизит формы `Объект`. То есть новые цены в табличной части теперь будут и в форме, которая пока находится на сервере.

Когда выполнение всех серверных процедур, которые мы вызвали, будет закончено, контекст формы на сервере будет собран, отправлен на клиент и там автоматически обновлен. В результате мы увидим в открытом документе на клиенте уже новые цены в табличной части документа.

Глава 3.8. Последовательность событий при открытии формы объекта

Теперь, когда мы знакомы с тем, как открыть нужную нам форму, и понимаем, как в принципе форма функционирует, рассмотрим подробно последовательность событий, которые возникают при открытии формы существующего объекта. Этот пример мы рассмотрим как наиболее сложный и типичный.

Ранее, на рис. 3.7, мы уже видели эту последовательность событий. Напомним (рис. 3.31).

Рис. 3.31. Последовательность событий при открытии формы существующего объекта

Теперь же мы можем представить ее более подробно (рис. 3.32).

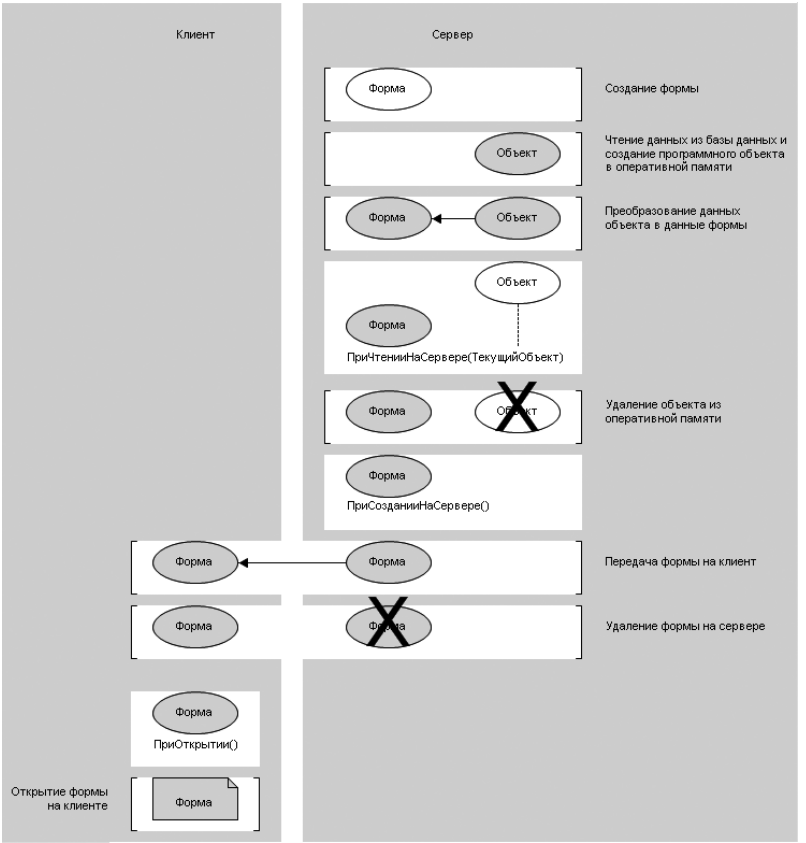


Рис. 3.32. Последовательность действий и событий при открытии формы существующего объекта

Чтение данных прикладного объекта

Когда платформа получает интерактивную или программную команду открыть форму объекта, прежде всего на сервере создается программный объект формы.

После этого создается программный объект, соответствующий типу основного реквизита формы, и из базы данных читаются данные этого объекта. На схеме (рис. 3.32) объект отмечен темным цветом. Это означает, что он содержит данные, прочитанные из базы данных.

Затем платформа выполняет преобразование данных объекта в данные формы. Об этом мы говорили в главе 3.4 на стр. 471.

Событие «При чтении на сервере»

После этого вызывается первое событие формы – При чтении на сервере.

В чем особенность этого события? Зачем именно в этот момент нам предлагается вмешаться в «процесс»?

Посмотрим. Событие вызывается у формы. Причем поставляется это событие не самой формой, а ее расширением, определяемым типом основного реквизита формы. То есть это какая-то особенная ситуация, связанная с тем, что открывается не просто форма, а именно форма этого объекта.

Данные формы уже заполнены данными объекта, в то же время сам объект все еще существует в памяти сервера и доступен в обработчике этого события через параметр ТекущийОбъект (листинг 3.26).

Листинг 3.26. Объявление обработчика события «При чтении на сервере»

```
&НаСервере
Процедура ПриЧтенииНаСервере(ТекущийОбъект)
```

Очевидно, что здесь мы можем подготовить дополнительные данные формы, которые зависят от данных объекта.

Существование самого объекта полезно тем, что мы можем воспользоваться всей его функциональностью, например вызвать какую-либо его экспортируемую процедуру или получить значение через точку от его реквизита ссылочного типа. Это избавляет нас от необходимости преобразовывать данные формы в объект.

После обработки события При чтении на сервере данные формы, вообще говоря, могут уже отличаться от тех данных, которые содержатся в прикладном объекте. Поэтому на схеме форма отмечена темным цветом. Это означает, что она содержит измененные данные.

Сам прикладной объект удаляется из памяти сервера, так как для дальнейших действий он уже не нужен. И вызывается второе и последнее событие формы на сервере – При создании на сервере.

Событие «При создании на сервере»

Что изменилось по сравнению с предыдущим событием? Нет прикладного объекта. Это последнее событие перед тем, как форма отправится на клиент. Это событие формы, то есть оно не зависит от того, какие данные форма отображает, и существует у всех форм.

Значит, в этом событии нужно полностью подготовить саму форму к открытию. Именно форму, ее внешнее представление. Данные мы уже полностью подготовили в предыдущем обработчике события.

Ранее мы уже использовали это событие, когда изучали различные способы открытия форм. Например, для того чтобы установить значения параметров произвольного запроса в динамическом списке.

Также в этом событии можно отказаться от открытия формы, если по каким-либо причинам она не должна быть открыта (листинг 3.27). Для этого параметр `Отказ` нужно установить в значение `Истина`.

Листинг 3.27. Объявление обработчика события «При создании на сервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
```

Также здесь можно отказаться от стандартной обработки события, выполняемой платформой (`СтандартнаяОбработка = Ложь`). Стандартная обработка, выполняемая платформой, зависит от типа основного реквизита открываемой формы.

Например, для формы списка это будет передача в динамический список параметров, указанных при открытии формы. Если в примере «Открыть список, чтобы курсор был на нужном элементе» в форме списка справочника `Товары` отменить стандартную обработку, то не будет выполняться позиционирование курсора на нужную строку в списке.

Аналогично, если в примере «Список подчиненного справочника с отбором по владельцу» в форме списка справочника `РасчетныеСчета` отменить стандартную обработку, то не будет выполняться отбор списка по владельцу.

Для форм отчетов стандартная обработка заключается в загрузке варианта и пользовательских настроек отчета. Кроме того, стандартная обработка передает в отчет параметры отбора и выполняет автоматический запуск отчета. Например, если в примере «Открытие и запуск отчета» в форме отчета ЦеныТоваров отменить стандартную обработку, то в отчет не будет передаваться отбор и не будет выполняться автоматическое формирование отчета.

Событие «При открытии»

После обработки события При создании на сервере форма передается на клиент, и программный объект формы удаляется из памяти сервера. На клиенте вызывается последнее событие формы – При открытии.

Для чего может понадобиться событие При открытии, обрабатываемое на клиенте?

Во-первых, это последнее событие, в котором можно отказаться от открытия формы (`Отказ = Истина`), листинг 3.28.

Листинг 3.28. Объявление обработчика события «При открытии»

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)
```

Если отказа не происходит, это значит, что форма точно будет открыта. И, значит, можно выполнить какие-то действия, которые должны выполняться только тогда, когда форма гарантированно открывается. Например, открыть связанную форму, которая сама по себе не существует, а существует только в том случае, если открыта основная форма.

Во-вторых, в этом событии можно выполнить те действия, которые невозможно выполнить на сервере. Например, выдать предупреждение пользователю или настроить COM-объект HTML-документа, содержащийся в поле HTML-документа.

После обработки события При открытии форма открывается на клиенте и становится видима пользователю.

Глава 3.9. Последовательность событий при записи объекта из формы

Последовательность событий, возникающих при записи объекта из формы, значительно больше и сложнее. Мы рассмотрим ее на примере команды Записать и закрыть, когда данные объекта записываются и после этого форма закрывается.

Поскольку схема получается довольно большой, разделим ее на три части:

- до записи данных в базу данных,
- запись данных,
- после записи.

Первая часть будет выглядеть следующим образом (рис. 3.33).

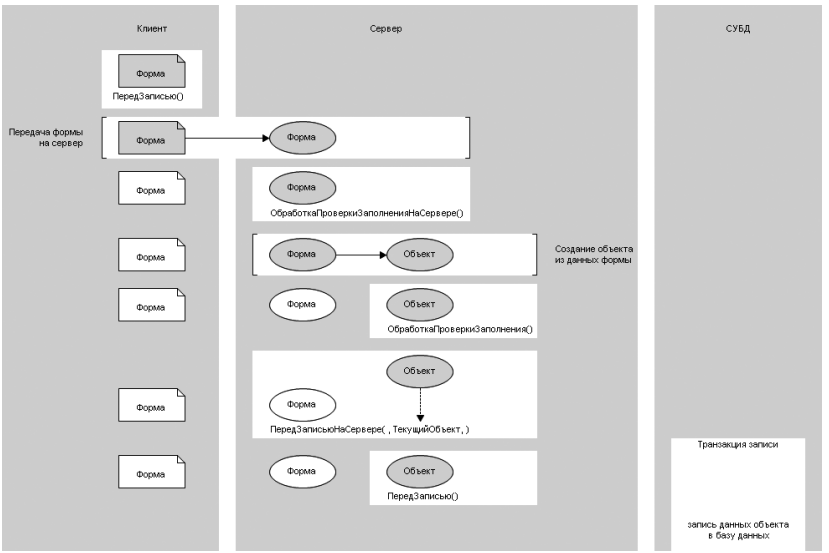


Рис. 3.33. Последовательность действий и событий при записи объекта. Начало

Событие «Перед записью»

Когда в форме вызывается запись объекта, прежде всего в форме на клиенте вызывается событие Перед записью.

Синтаксис вызова этого обработчика выглядит следующим образом (листинг 3.29).

Листинг 3.29. Объявление обработчика события «Перед записью»

```
&НаКлиенте
Процедура ПередЗаписью(Отказ, ПараметрыЗаписи)
```

В этом обработчике можно проанализировать, все ли вспомогательные данные, необходимые для записи объекта, подготовлены. И если это не так – отказаться от записи, установив параметр `Отказ` в значение `Истина`. Дело в том, что на клиенте могут существовать объекты, недоступные на сервере, например COM-объект HTML-документа. Поэтому проверки, связанные с подобными объектами, нужно производить именно в клиентской процедуре.

Также в этом обработчике доступны для анализа *параметры записи*, созданные платформой или переданные разработчиком. Подробнее о параметрах записи написано в разделе «Параметры записи» на стр. 510.

Проверка заполнения

После обработки события Перед записью платформа выполняет контекстный серверный вызов. Контекст формы упаковывается, передается на сервер. На сервере создается программный объект формы. Из полученного контекста заполняются реквизиты, элементы, параметры формы, инициализируется модуль формы, и в дело вступает *механизм проверки заполнения*.

Его работа заключается в последовательном вызове двух событий: одного – у формы, второго – у того объекта, который будет записан. Работа механизма проверки заполнения подробно рассматривается в главе 3.11 на стр. 525, поэтому сейчас мы не будем подробно останавливаться на назначении каждого из событий. Опишем их лишь в общих чертах.

Сначала вызывается событие формы Обработка проверки заполнения на сервере.

После этого выполняется преобразование данных формы в данные прикладного объекта: в памяти сервера создается прикладной объект, соответствующий типу основного реквизита формы, и его данные заполняются из данных формы.

Затем вызывается событие прикладного объекта Обработка проверки заполнения.

Событие «Перед записью на сервере»

После того как механизм проверки заполнения закончил свою работу, на сервере вызывается событие формы Перед записью на сервере.

Чем интересен этот обработчик? Как мы говорили выше, в процессе проверки заполнения произошло «разделение» формы на форму и прикладной объект, данные которого будут записаны в базу данных. Это определяет ряд особенностей дальнейшей работы.

Обработчик Перед записью на сервере – первый, в котором появляется возможность доступа как к данным основного реквизита формы (через его имя, как правило – `Объект`), так и к самому объекту, который будет записан. Этот объект система передает в обработчик в параметре `ТекущийОбъект`. Напомним описание вызова этого обработчика (листинг 3.30).

Листинг 3.30. Объявление обработчика события «Перед записью на сервере»

```
Процедура ПередЗаписьюНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)
```

Что важно понимать, находясь в этом обработчике? Данные, доступные через основной реквизит формы `Объект`, «мертвы». Мертвы в том смысле, что их изменение не даст никакого результата. Их можно только анализировать.

При этом также нужно понимать, что эти данные – не то, что будет записано в информационную базу. Это то, что «приехало» с клиента и «пережило» проверку заполнения. Поэтому и анализировать их можно только с точки зрения того, «а что же было».

А записан в информационную базу будет ТекущийОбъект. Отсюда еще два замечания:

- Если нужно принять решение о том, можно записывать данные или нельзя, нужно анализировать ТекущийОбъект.
- Если нужно модифицировать записываемые данные (например, дописать служебную информацию в реквизиты), использовать нужно ТекущийОбъект.

Все попытки изменить данные через реквизит формы Объект ни к чему не приведут. В информационную базу эти данные записаны не будут, а перед тем как форма «поедет» обратно на клиент, ее данные будут замещены данными записанного в информационную базу объекта. Таким образом, все изменения, внесенные в Объект, пропадут.

Также нужно понимать, что все действия, выполняемые в этом обработчике (отказ от записи, изменение данных объекта), должны быть связаны именно с тем фактом, что данные записываются именно из формы. При программной записи объекта (методом объекта) форма будет отсутствовать, следовательно, и это событие вызываться не будет.

Поэтому если какие-либо алгоритмы должны выполняться при любом способе записи данных объекта, а не только при записи из формы, их следует размещать в обработчике события объекта (Перед записью), а не в обработчике события формы.

Запись данных в базу данных

После обработки события формы Перед записью на сервере в СУБД открывается транзакция записи и начинается процесс записи данных прикладного объекта в базу данных, который представлен на рисунке 3.34.

Прежде всего вызывается событие объекта Перед записью. В этом событии можно отказаться от записи данных. Причем эта проверка будет выполняться всегда, независимо от того, каким способом данные объекта записываются.

После обработки события Перед записью данные записываются в базу данных, но транзакция не закрывается.

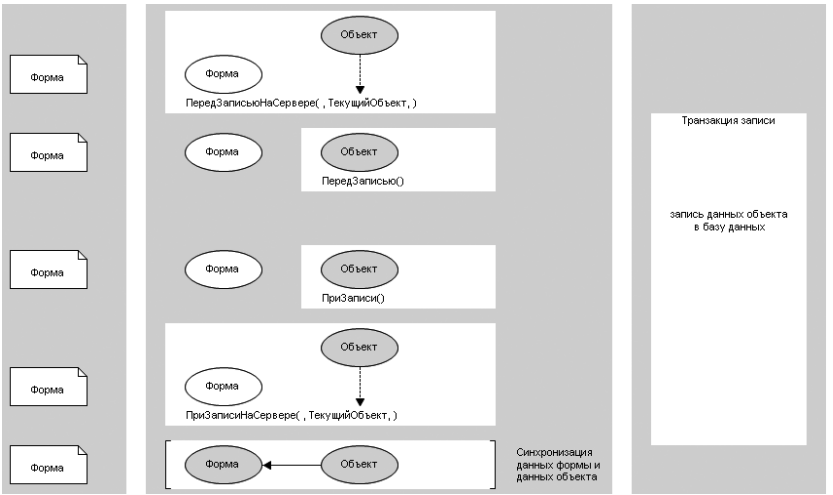


Рис. 3.34. Последовательность действий и событий при записи объекта. Продолжение

После записи данных вызывается еще одно событие объекта – При записи. Это событие предназначено для того, чтобы выполнить действия над дополнительными данными, которые неразрывно связаны с основными данными объекта и не могут быть изменены независимо от основных данных. Поскольку это событие обрабатывается в транзакции записи основных данных, гарантируется синхронность изменения основных и вспомогательных данных. Либо и те и другие будут записаны, либо и те и другие изменения будут отменены при отмене транзакции.

После обработки события объекта При записи, пока транзакция открыта, на сервере вызывается событие формы При записи на сервере. Назначение этого обработчика аналогично предыдущему – чтобы записать в базу данных дополнительную информацию, связанную с данными записываемого объекта. Ведь совсем не обязательно, что все исходные данные для записи дополнительной информации находятся в самом объекте. Они могут находиться и в форме. Как раз для таких случаев и предназначено это событие.

Описание процедуры, обрабатывающей это событие, выглядит следующим образом (листинг 3.31).

Листинг 3.31. Объявление обработчика события «При записи на сервере»

```
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)
```

Находясь в этом обработчике, важно понимать, что имеет смысл работать только с теми данными объекта, которые доступны через `ТекущийОбъект`. Здесь `ТекущийОбъект` – это именно то, что было записано в информационную базу.

Те данные, которые доступны через основной реквизит формы `Объект`, – это «старые» данные, то, что было в форме, когда она «приехала» на сервер. Изменять эти данные бесполезно, потому что после выхода из этого обработчика они будут заменены данными, содержащимися в `ТекущийОбъект`.

Иногда имеет смысл их анализировать. Например, в случае записи нового объекта тут будет следующая ситуация. `Объект.Ссылка`, скорее всего, будет содержать пустую ссылку (т. к. пока новый объект не записан, у него нет ссылки). В то же время `ТекущийОбъект.Ссылка` будет содержать уже конкретное значение ссылки на этот элемент в информационной базе. Конечно, все это справедливо только в том случае, если не выполнялось программное изменение ссылки до или после записи.

Итак, по поводу этого обработчика можно сделать следующие замечания:

- Если нужно выполнять какие-то действия, связанные с записанным объектом, и при этом, например, нужна ссылка на этот объект, необходимо использовать `ТекущийОбъект.Ссылка`.
- Если нужно изменить записанные данные перед отправкой их на клиент, необходимо использовать `ТекущийОбъект`.
- Основной реквизит формы `Объект` можно использовать только для сравнения того, что «было», с тем, что «записалось». Изменять его бессмысленно, т. к. он будет замещен данными из `ТекущийОбъект`.

Событие «После записи на сервере»

После того как обработано событие формы При записи на сервере, транзакция записи в СУБД закрывается и начинается процесс передачи данных на клиент и закрытия формы. Он выглядит следующим образом (рис. 3.35).

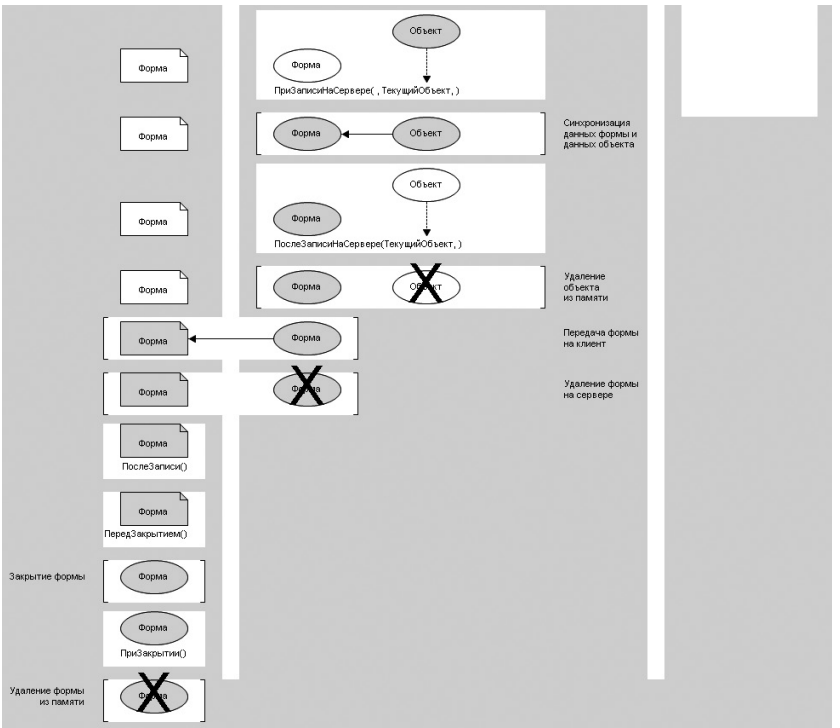


Рис. 3.35. Последовательность действий и событий при записи объекта. Окончание

После завершения транзакции записи выполняется преобразование данных записанного объекта в данные формы. После этого вызывается событие формы После записи на сервере. Это третий и последний обработчик, в котором по отдельности доступны данные формы и объект, который был записан. Вызов этого обработчика также содержит `ТекущийОбъект` и похож на предыдущие (листинг 3.32).

Листинг 3.32. Объявление обработчика события «После записи на сервере»

```
Процедура ПослеЗаписиНаСервере(ТекущийОбъект, ПараметрыЗаписи)
```

Однако здесь назначение `ТекущийОбъект` – прямо противоположное тому, что было до этого. Объясним подробнее.

В этом обработчике данные записанного объекта уже помещены в форму. Поэтому «живым» в этом обработчике является уже основной реквизит формы `Объект`. А `ТекущийОбъект` существует лишь для того, чтобы дать возможность разработчику выполнить какие-то вспомогательные действия.

Что обычно делают в этом обработчике? Обработчик – в модуле формы, значит, предполагаются действия над формой. Обработчик вызывается после окончания транзакции записи – значит, это действия, которые должны быть выполнены только в том случае, когда объект 100 % записан. Например, вывод в форме некоторой дополнительной информации, связанной с основными данными объекта. Или выполнение каких-либо действий, которые должны быть выполнены только в том случае, когда объект гарантированно записан.

Отсюда последние замечания:

- `ТекущийОбъект` изменять бессмысленно: он будет уничтожен при выходе из обработчика.
- Если для выполнения связанных действий нужны какие-то данные или методы объекта, нужно использовать `ТекущийОбъект`, а не пытаться получать их через основной реквизит формы.

Передача формы на клиент

После обработки события После записи на сервере форма передается на клиент. В памяти сервера уничтожается прикладной объект. Измененные реквизиты, элементы и параметры формы, требуемые для отображения, передаются на клиент. Уничтожается объект формы на сервере. После этого на клиенте вызывается событие формы После записи.

Событие «После записи»

В обработчике этого события также выполняются действия над формой, которые должны быть выполнены только в случае 100 % записи данных объекта. Те действия, которые требуют интерактивного взаимодействия с пользователем или которые невозможно выполнить на сервере.

Событие «Перед закрытием»

Так как мы рассматриваем ситуацию записи объекта и закрытия его формы, то после обработки на клиенте события После записи на клиенте же вызывается событие Перед закрытием.

Основное назначение этого события – проанализировать, можно закрывать форму или нельзя. И если нельзя, то отказаться от ее закрытия. Описание процедуры обработчика этого события выглядит следующим образом (листинг 3.33).

Листинг 3.33. Объявление обработчика события «Перед закрытием»

```
&НаКлиенте
Процедура ПередЗакрытием(Отказ, ЗавершениеРаботы, ТекстПредупреждения, СтандартнаяОбработка)
```

Установка параметра Отказ в значение Истина позволяет отказаться от закрытия формы. А запрет стандартной обработки (СтандартнаяОбработка = Ложь) позволяет отказаться от стандартных действий, которые опять-таки определяются типом основного реквизита формы.

Например, в случае с формой справочника стандартные действия платформы будут заключаться в следующем. Если форма модифицирована и окно формы закрывается клавишей Esc или нажатием кнопки закрытия окна, то платформа сообщит о том, что данные формы модифицированы, и предложит либо записать их, либо закрыть без сохранения изменений, либо отказаться от закрытия формы.

Если в этой ситуации отказаться от стандартной обработки, то форма будет закрыта без вопросов в любом случае, независимо от того, модифицированы в ней данные или нет.

С помощью параметра ЗавершениеРаботы разработчик может определить, закрывается форма «сама по себе» или форма закрывается в процессе завершения работы приложения. Данный параметр устанавливается в значение Истина в том случае, если пользователь пытается закрыть основное окно приложения. При закрытии отдельной формы данный параметр будет установлен в значение Ложь.

При закрытии всего приложения и при отказе от закрытия конкретной формы с помощью параметра ТекстПредупреждения разработчик может установить текстовое сообщение, которое будет показано пользователю, с возможностью отказаться от закрытия или подтвердить его (в веб-клиенте все сообщения будут объединены в один диалог).

Если параметр ТекстПредупреждения не установлен, то в качестве текста предупреждения в диалоге будет показана строка «Работа в данном окне не завершена».

Нужно помнить, что при обработке завершения работы приложения в обработчиках событий ПередЗавершениемРаботыСистемы, ПриЗавершенииРаботыСистемы, а также в обработчиках событий формы, находящейся в режиме закрытия (ПередЗакрытием, ПриЗакрытии), запрещено открывать окна и выполнять любые серверные вызовы. Поэтому рекомендуется заранее сохранить в данных формы ту информацию, которая может потребоваться, чтобы определить, можно закрыть форму при завершении работы приложения или нет.

Как уже говорилось, при закрытии форм прикладных объектов (по отдельности или в случае закрытия основного окна приложения), если данные были изменены, платформа предупредит об этом и не даст закрыть окно без подтверждения пользователя. Это поведение не требует вмешательства разработчика, в случае если стандартная обработка события ПередЗакрытием не отменена.

В противном случае, или если форма произвольная (т. е. прикладной объект не является основным реквизитом формы), или это, например, форма обработки, разработчик должен самостоятельно реализовать принятый в платформе единый подход к закрытию форм и основного окна приложения.

В обработчике события ПередЗакрытием по значению параметра ЗавершениеРаботы разработчик должен определить, закрывается ли форма в процессе завершения работы приложения. Если да, то разработчик, проанализировав данные формы, принимает решение, нужно ли выдать пользователю какое-либо сообщение, требующее реакции пользователя: завершить работу или все-таки продолжить работу, т. к. в форме есть существенная информация.

- Если разработчик считает, что пользователю необходимо выдать сообщение, то он задает этот текст в значении параметра ТекстПредупреждения и устанавливает параметр Отказ в значение Истина.
- В результате при завершении работы приложения текст предупреждения, который задал разработчик, будет показан пользователю с вариантами ответа: Завершить работу или Продолжить работу.

Если параметр ЗавершениеРаботы имеет значение Ложь, то это означает, что закрывается отдельная форма. В этом случае логика работы выглядит следующим образом:

- Вначале задается вопрос о необходимости закрытия формы, а собственно закрытие формы отменяется.
- После того как пользователь ответил на вопрос, в специальной клиентской переменной отмечается, что сейчас будет выполнено «настоящее» закрытие формы, и форма закрывается повторно.

Поясним вышесказанное на примере.

Пример можно посмотреть в демонстрационной базе «Открытие форм», форма обработки ЗакрытиеФормы.

В форме обработки ЗакрытиеФормы содержится булевый реквизит ПредупреждениеОЗакрытии, который отображается в форме в виде флажка. Для упрощения примера вместо «настоящего» анализа данных формы мы будем использовать состояние этого флажка в качестве признака того, надо ли показывать пользователю предупреждение с вопросом о возможности закрытия формы или закрывать ее сразу, без вопросов.

Рассмотрим обработчик события формы ПередЗакрытием (листинг 3.34).

Листинг 3.34. Обработчик события «Перед закрытием»

```
&НаКлиенте
Процедура ПередЗакрытием(Отказ, ЗавершениеРаботы, ТекстПредупреждения, СтандартнаяОбработка)
    Если НЕ ПредупреждениеОЗакрытии Тогда
        Возврат;
    КонецЕсли;

    Если НЕ ЗавершениеРаботы Тогда
        Если ОтветПередЗакрытием <> Истина Тогда
            Отказ = Истина;
            ОповещениеОЗакрытии = Новый ОписаниеОповещения(
                "ПередЗакрытиемЗавершение", ЭтотОбъект);
            ПоказатьВопрос(ОповещениеОЗакрытии, "Вы хотите закрыть форму?",
                РежимДиалогаВопрос.ДаНет);
        КонецЕсли;
    Иначе
        Отказ = Истина;
        ТекстПредупреждения = "При закрытии формы все данные будут утеряны. Продолжить?";
    КонецЕсли;
КонецПроцедуры
```

Прежде всего, если флажок ПредупреждениеОЗакрытии не установлен, то ничего не происходит и форма закрывается без вопросов.

В противном случае, если параметр ЗавершениеРаботы имеет значение Истина, то форма закрывается в процессе завершения работы приложения.

В этом случае параметр Отказ мы устанавливаем в значение Истина и задаем в параметре ТекстПредупреждения сообщение, которое будет показано пользователю перед закрытием формы обработки.

Если параметр ЗавершениеРаботы имеет значение Ложь, значит, форма обработки закрывается, как обычно, кнопкой Закрыть.

В этом случае мы проверяем значение клиентской переменной модуля формы ОтветПередЗакрытием. Если это значение истинно, значит, ответ пользователя на вопрос о возможности закрытия формы уже был получен, и форма действительно закрывается. Если значение переменной ОтветПередЗакрытием не определено, значит, вопрос перед закрытием формы пользователю еще не задавался.

В этом случае параметр Отказ мы устанавливаем в значение Истина и создаем описание оповещения, определяющее процедуру, которая будет выполнена, после того как пользователь ответит на вопрос.

Вопрос о возможности закрытия формы задаем пользователю с помощью немодального метода ПоказатьВопрос() и передаем туда первым параметром описание оповещения.

Процедура обработки оповещения ПередЗакрытиемЗавершение() выглядит следующим образом (листинг 3.35).

Листинг 3.35. Процедура «ПередЗакрытиемЗавершение»

```
&НаКлиенте
Процедура ПередЗакрытиемЗавершение(Результат, ДополнительныеПараметры) Экспорт
        Если Результат = КодВозвратаДиалога.Да Тогда
                ОтветПередЗакрытием = Истина;
                Закрыть();
        Иначе
                ОтветПередЗакрытием = Неопределено;
        КонецЕсли;
КонецПроцедуры
```

В процедуре обработки оповещения, в случае если пользователь согласился с закрытием формы, переменная ОтветПередЗакрытием устанавливается в значение Истина, и форма закрывается еще раз методом Закрыть(). Если нет, то форма остается открытой и переменная ОтветПередЗакрытием устанавливается в значение Неопределено.

Таким образом, мы задаем пользователю вопрос о возможности закрытия формы в обработчике события ПередЗакрытием. Обработчик вызывается дважды. Первый раз – когда закрытие формы (отдельно или при завершении работы приложения) инициируется пользователем, а второй – из встроенного языка после подтверждения пользователя. Но вопрос задается пользователю только в первый раз, если переменная ОтветПередЗакрытием не определена.

Остается только объявить эту клиентскую переменную в модуле формы (листинг 3.36).

Листинг 3.36. Объявление клиентской переменной «ОтветПередЗакрытием»

```
&НаКлиенте
Перем ОтветПередЗакрытием;
```

В результате в режиме 1С:Предприятие при закрытии формы обработки кнопкой Закрыть и при установленном флажке Предупре-

ждение о закрытии пользователю будет показан следующий диалог, в котором он может подтвердить закрытие формы или отказаться от него (рис. 3.36).

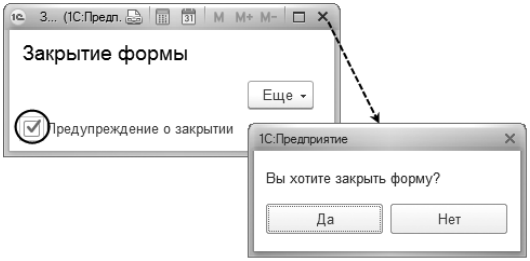


Рис. 3.36. Вопрос пользователю перед закрытием формы обработки

А при закрытии формы обработки в процессе завершения работы приложения пользователю будет показан следующий диалог (рис. 3.37).

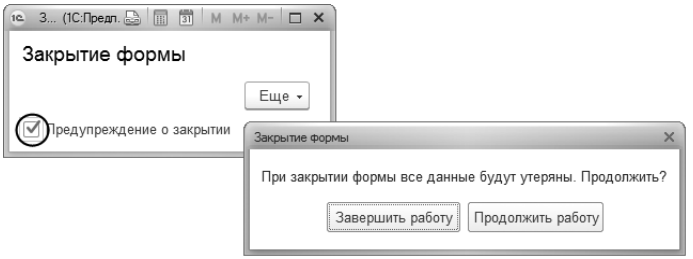


Рис. 3.37. Вопрос пользователю перед закрытием формы обработки

Событие «При закрытии»

После обработки события Перед закрытием форма закрывается, пользователь перестает ее видеть. Однако программный объект формы все еще существует, и на клиенте вызывается событие формы При закрытии.

Назначение этого события в том, чтобы выполнить действия, которые должны быть выполнены только в случае 100 % закрытия формы.

Например, закрыть вспомогательную форму, существующую только в том случае, если открыта основная форма.

В обработчик события ПриЗакрытии в параметре ЗавершениеРаботы также передается признак того, при каких условиях закрывается форма. Но надо иметь в виду, что отказаться от закрытия формы в данном обработчике нельзя.

После обработки этого события программный объект формы удаляется из памяти клиента.

Параметры записи

Запись данных объекта в форме может быть выполнена средствами встроенного языка. Для этого у многих расширений формы существует метод `Записать()`. У этого метода есть единственный параметр – `ПараметрыЗаписи`. Он содержит структуру произвольных параметров записи.

Некоторые расширения форм объектов добавляют собственные, *предопределенные параметры записи*. Например, расширение документа добавляет параметры `РежимЗаписи` и `РежимПроведения`, расширение задачи – параметр `ВыполнитьЗадачу`.

Подробнее о предопределенных параметрах можно прочитать в синтакс-помощнике, в описании события Перед записью для разных расширений формы или в описании других событий, перечисленных ниже.

Наряду с предопределенными параметрами разработчик может добавлять и *собственные параметры*. Эти параметры разработчик может использовать в различных обработчиках событий записи в зависимости от своих нужд.

Таких событий, в которых будут доступны параметры записи, пять. На рисунке 3.38 они выделены рамкой.

В любом из этих обработчиков параметры записи можно изменять, добавлять, удалять – все выполненные изменения будут переданы в следующий обработчик.

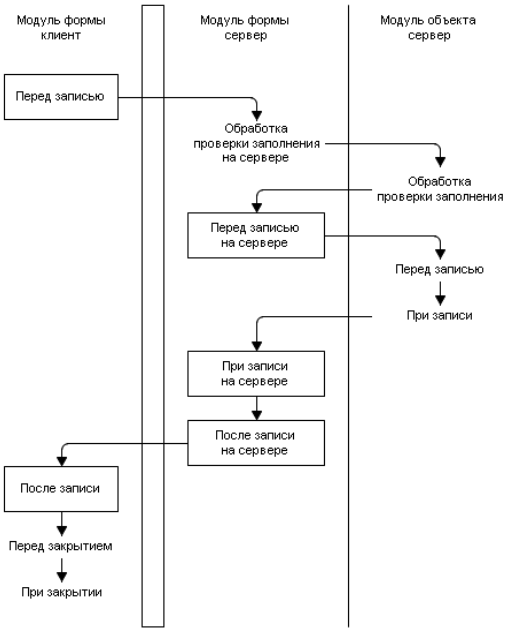


Рис. 3.38. События, в которых доступны параметры записи

Глава 3.10. Начальное заполнение

Механизм начального заполнения позволяет описать правила и алгоритмы, по которым новые объекты, интерактивно создаваемые в информационной базе, будут заполнены некоторыми данными.

Механизм начального заполнения ориентирован исключительно на интерактивную работу. При создании новых объектов средствами встроенного языка он не вызывается. Но при необходимости разработчик может его задействовать, для того чтобы имитировать интерактивное создание новых объектов.

Начальное заполнение вызывается в следующих случаях интерактивного создания нового объекта:

- командой Создать в панели функций текущего раздела или в списке;
- командой ввода на основании;
- при программном вызове методов глобального контекста `ОткрытьФорму()` или `ПолучитьФорму()`;
- при программном вызове методов объектов `Заполнить()`.

Кроме перечисленных способов есть еще один способ интерактивного создания нового объекта – копированием. Этот случай обрабатывается платформой отдельно, и механизм начального заполнения при этом не задействуется.

Механизм начального заполнения имеет несколько составляющих, часть из них доступна в режиме Конфигуратор, часть – во встроенном языке в режиме 1С:Предприятие:

- Во-первых, в конфигураторе можно задать конкретные значения, которыми автоматически будут заполнены реквизиты нового объекта. У реквизитов есть свойство Значение заполнения.
- Во-вторых, в конфигураторе можно разрешить платформе в некоторых случаях самостоятельно заполнить реквизиты нового объекта подходящими данными. У реквизитов есть свойство Заполнять из данных заполнения.
- В-третьих, во встроенном языке разработчик может описать собственные алгоритмы заполнения реквизитов данными в обра-

ботчике события объекта Обработка заполнения. Это событие вызывается в тех случаях, которые были перечислены выше у прикладных объектов (справочников, документов и т. д.) и у набора записей регистра сведений.

Рассмотрим каждый из этих трех элементов подробнее. В качестве примера мы будем использовать демонстрационную базу «Начальное заполнение».

Свойство «Значение заполнения»

У реквизитов объектов конфигурации существует свойство Значение заполнения.

Если реквизит примитивного типа (Строка, Число, Дата, Булево), то в свойстве можно задать любое значение этого типа, которым платформа автоматически заполнит этот реквизит у нового объекта.

Если реквизит имеет тип ссылки или перечисления, то в этом свойстве можно задать предопределенный элемент или конкретное значение перечисления.

Например, у документа Накладная может быть реквизит Склад с типом ссылки на справочник Склады. А в справочнике Склады может существовать предопределенный элемент Главный склад.

Пример можно посмотреть в демонстрационной базе «Начальное заполнение», документ Накладная, реквизит Склад.

Тогда прямо в конфигураторе можно указать, что в новых документах реквизит Склад будет сразу же заполнен ссылкой на главный склад (рис. 3.39).

В демонстрационной базе можно создать новую накладную (командой в панели функций текущего раздела или командой в списке) и убедиться, что склад в новой накладной заполнен (рис. 3.40).

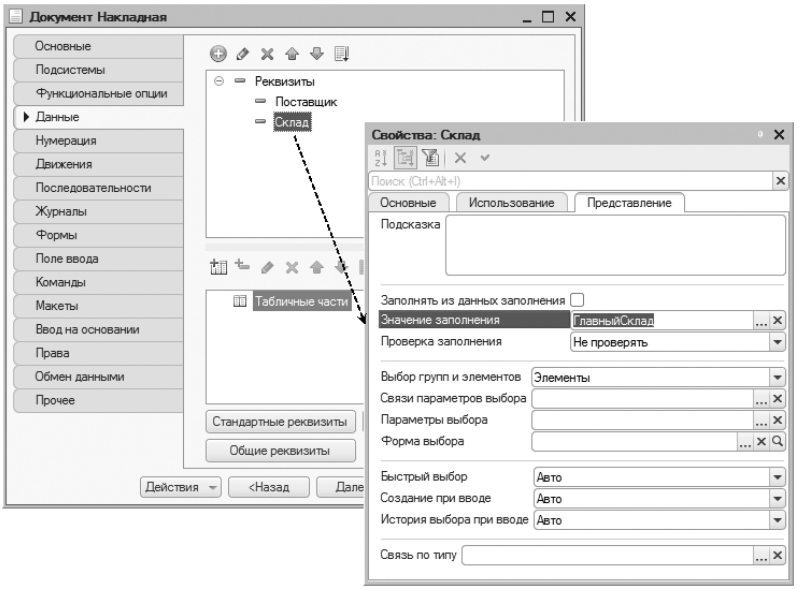


Рис. 3.39. Свойство «Значение заполнения»

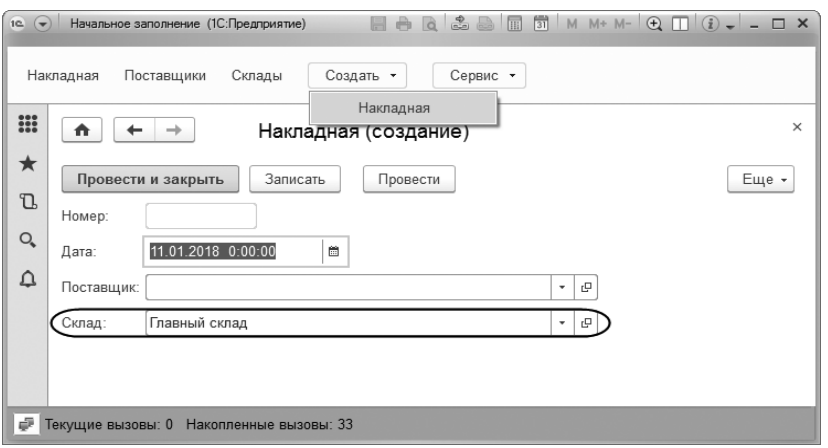


Рис. 3.40. Автоматическое заполнение реквизита для новых объектов

Свойство «Заполнять из данных заполнения»

Другая возможность обеспечить начальное заполнение данных нового объекта – это разрешить платформе в некоторых ситуациях сделать это автоматически. Для этого используется другое свойство реквизитов объекта конфигурации – Заполнять из данных заполнения.

Если это свойство установлено, платформа автоматически заполнит значение такого реквизита из *данных заполнения* в том случае, если они представляют собой структуру, содержащую отборы, и имя одного из условий отбора совпадает с именем этого реквизита.

Вообще в данные заполнения платформа может передавать различные типы значений.

Например, если просто создается новый объект, то в данных заполнения будет значение `Неопределено`. Если новый объект создается вводом на основании, то в данных заполнения будет ссылка на объект, являющийся основанием. Но во всех этих случаях платформа не будет автоматически заполнять реквизиты из данных заполнения.

Автоматическое заполнение будет только в том случае, когда данные заполнения содержат структуру, состоящую из условий отбора.

Когда в данных заполнения может оказаться такая структура?

- Во-первых, когда новый объект вводится командой Создать из списка, в котором установлены некоторые отборы. Тогда все эти отборы платформа автоматически поместит в данные заполнения.
- Во-вторых, когда новый объект вводится в результате программного вызова методов `ОткрытьФорму()`, `ПолучитьФорму()` или метода объекта `Заполнить()`. Во все эти методы разработчик самостоятельно может передать структуру, содержащую нужные отборы.

Рассмотрим оба этих случая.

Создание объекта из отобранного списка

Пример можно посмотреть в демонстрационной базе «Начальное заполнение», документ Накладная, реквизит Поставщик.

Сначала рассмотрим случай, когда, например, в списке накладных установлен отбор и пользователь нажимает кнопку Создать в командной панели списка. При этом у накладной существует реквизит Поставщик, для которого установлено свойство Заполнять из данных заполнения (рис. 3.41).

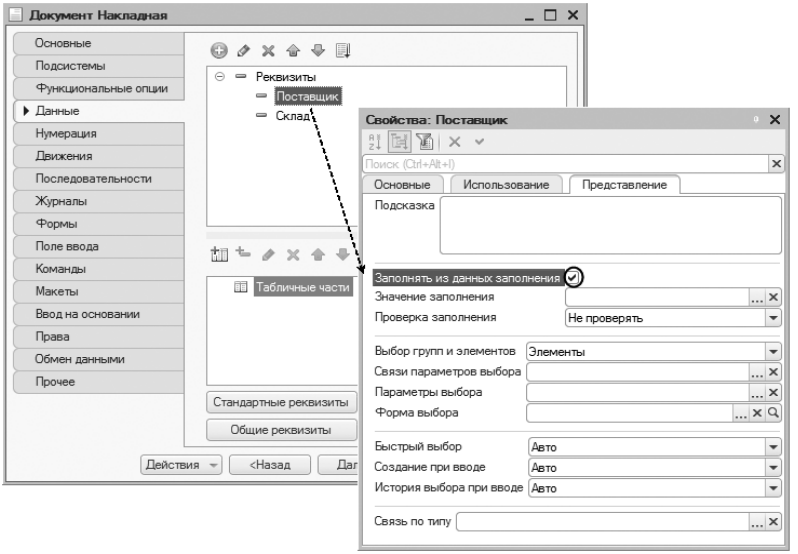


Рис. 3.41. Свойство «Заполнять из данных заполнения»

В режиме 1С:Предприятие в списке накладных установлены отборы по поставщику и по складу (рис. 3.42).

Командой Создать списка можно создать новую накладную и убедиться, что поставщик будет заполнен тем значением, по которому был установлен отбор – Поставщик 3 (рис. 3.43).

При этом склад будет по-прежнему заполнен значением Главный склад, т. к. в конфигураторе оно указано как значение заполнения.

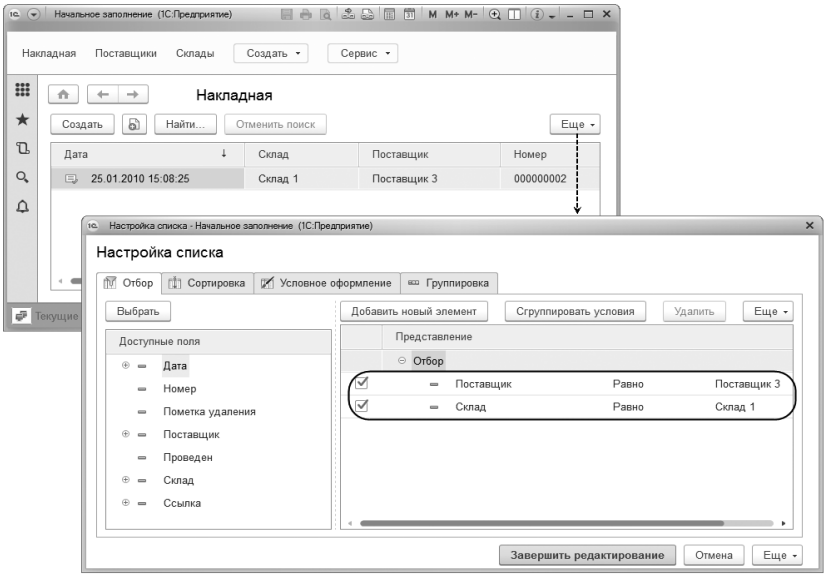


Рис. 3.42. Отбор, установленный в списке накладных

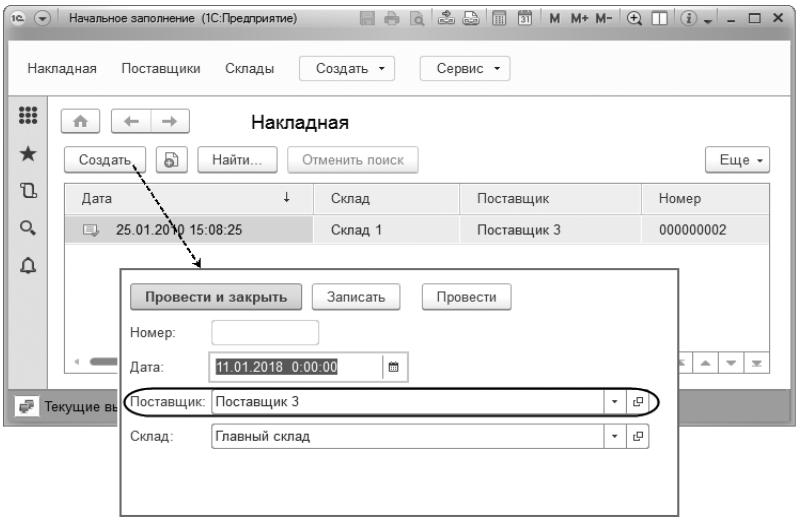


Рис. 3.43. Автоматическое заполнение реквизитов данными отбора

Если в конфигураторе для реквизита Склад тоже установить свойство Заполнять из данных заполнения, то тогда при создании из отобранного списка и поставщик, и склад будут заполнены значениями отбора, несмотря на то что для склада установлено значение заполнения (рис. 3.44).

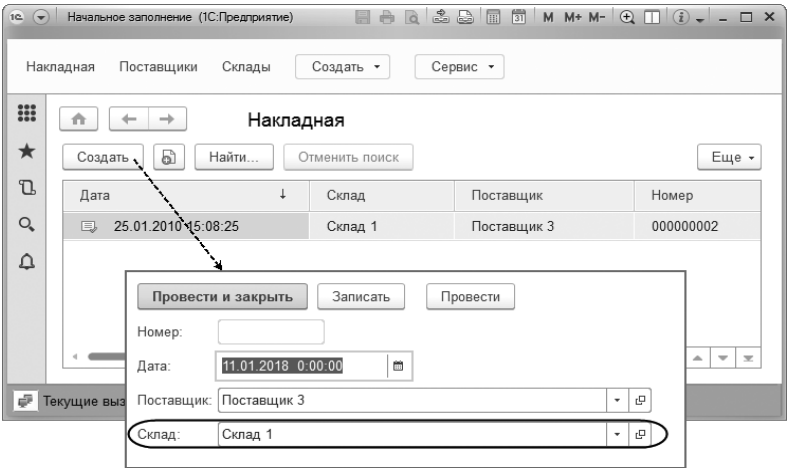


Рис. 3.44. Приоритет данных заполнения перед значением заполнения

Программная установка данных заполнения

Теперь рассмотрим второй случай, когда данные заполнения устанавливаются программно.

Например, выполняется метод встроенного языка `ОткрытьФорму()`. Как говорилось выше, при программном открытии формы есть возможность задать значения параметров формы. Для решения нашей задачи предназначен параметр расширения формы `ЗначенияЗаполнения`.

Например, мы хотим открыть форму и сразу же указать, что реквизит Поставщик должен быть заполнен некоторой конкретной ссылкой.

> Пример можно посмотреть в демонстрационной базе «Начальное заполнение», глобальная команда Открыть форму.

Для этого создадим серверную функцию, которая вернет эту самую ссылку на поставщика (листинг 3.37).

Листинг 3.37. Серверная функция, возвращающая ссылку на поставщика

```
&НаСервере
Функция ПолучитьПоставщикаНаСервере()

    Возврат Справочники.Поставщики.НайтиПоКоду("000000001");

КонецФункции
```

А затем в клиентском обработчике команды Открыть форму создадим структуру, содержащую отбор по поставщику (`ЭлементыОтбора`), и передадим ее в параметр формы `ЗначенияЗаполнения` (листинг 3.38).

Листинг 3.38. Установка значения заполнения при открытии формы

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

    СсылкаНаПоставщика = ПолучитьПоставщикаНаСервере();
    ЭлементыОтбора = Новый Структура("Поставщик", СсылкаНаПоставщика);
    ПараметрыФормы = Новый Структура("ЗначенияЗаполнения", ЭлементыОтбора);
    ОткрытьФорму("Документ.Накладная.ФормаОбъекта", ПараметрыФормы);

КонецПроцедуры
```

В результате выполнения такой команды будет открыта форма новой накладной, в которой поставщик будет заполнен значением `Поставщик 1` (рис. 3.45).

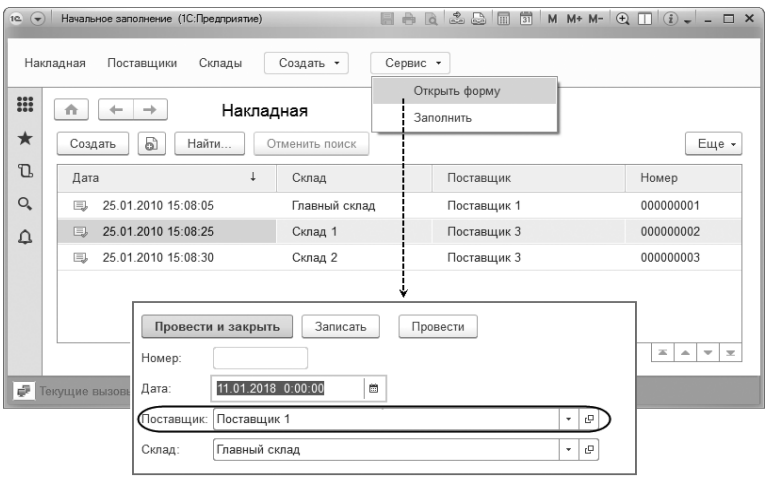


Рис. 3.45. Результат программной установки значения заполнения

В ситуации, когда новый объект полностью создается и записывается без участия пользователя, можно использовать метод объекта Заполнить(), чтобы смоделировать создание объекта по тем правилам и алгоритмам, которые используются при интерактивном создании.

Пример можно посмотреть в демонстрационной базе «Начальное заполнение», глобальная команда Заполнить.

Например, находясь на сервере, можно создать новый объект накладной и затем вызвать у него метод Заполнить(). В этот метод передать значения, необходимые для начального заполнения, например поставщика.

При интерактивном создании накладной расширение формы автоматически устанавливает дату нового документа. В программном создании накладной форма не участвует, поэтому в структуре данных заполнения нужно передать еще и значение для поля Дата, иначе записать такую накладную не удастся. А для того чтобы платформа обработала его автоматически, в конфигураторе у стандартного реквизита накладной Дата нужно установить свойство Заполнять из данных заполнения (листинг 3.39).

Листинг 3.39. Использование метода «Заполнить()»

```
&НаСервере
Процедура СоздатьИЗаписатьОбъектНаСервере()

    ССылкаНаПоставщика = Справочники.Поставщики.НайтиПоКоду("000000002");
    ЭлементыОтбора = Новый Структура("Поставщик, Дата", ССылкаНаПоставщика, ТекущаяДата());

    ОбъектНакладной = Документы.Накладная.СоздатьДокумент();
    ОбъектНакладной.Заполнить(ЭлементыОтбора);
    ОбъектНакладной.Записать();

КонецПроцедуры
```

В результате выполнения команды Заполнить (из которой вызывается процедура СоздатьИЗаписатьОбъектНаСервере()) будет создана и записана новая накладная с текущей датой и поставщиком Поставщик 2. Значение для реквизита Склад будет взято из его свойства Значение заполнения (рис. 3.46).

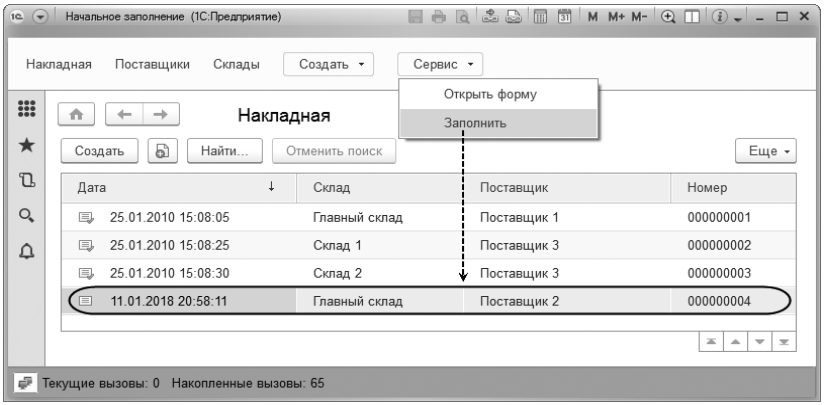


Рис. 3.46. Результат использования метода «Заполнить()»

Следует заметить, что для стандартных реквизитов Родитель, Владелец и для ведущих измерений регистров сведений платформа автоматически в конфигураторе устанавливает свойство Заполнять из данных заполнения. Для остальных стандартных реквизитов, как в нашем случае для даты, нужно это свойство устанавливать самостоятельно, если планируется использовать возможность их автоматического заполнения.

Событие «Обработка заполнения»

Если разработчика не устраивают значения заполнения, установленные в конфигураторе, или значения, переданные в данные заполнения, он может реализовать собственный алгоритм заполнения нового объекта данными. Для этого у объекта есть событие Обработка заполнения, которое может быть обработано в серверной процедуре.

Чтобы лучше представлять себе возможности программной обработки заполнения, рассмотрим еще раз последовательность, в которой платформа использует те или иные элементы механизма заполнения (рис. 3.47).

Сначала будет вызван обработчик события Обработка заполнения и выполнен его код.

Если после выхода из этого обработчика его параметр `СтандартнаяОбработка` будет иметь значение `Ложь`, то платформа не будет

пытаться самостоятельно заполнить реквизиты нового объекта, а сразу же вызовет событие формы При создании на сервере.

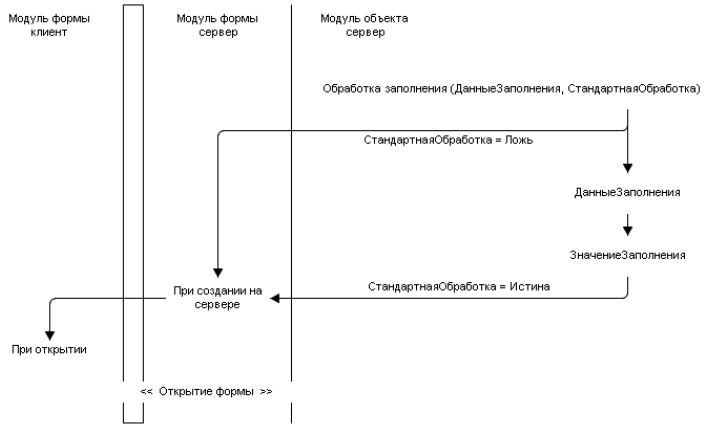


Рис. 3.47. Последовательность автоматического заполнения реквизитов нового объекта

Если же после выхода из обработчика значение его параметра `СтандартнаяОбработка` будет `Истина` (по умолчанию), то сначала платформа попытается заполнить значения реквизитов из данных заполнения.

Если после этого реквизит все еще будет иметь значение своего типа по умолчанию, то платформа попытается заполнить его значением заполнения, указанным в конфигураторе. Если же после данных заполнения реквизит будет иметь значение, отличное от значения по умолчанию, то значение заполнения использоваться не будет, даже если оно указано для этого реквизита.

Таким образом, находясь в обработчике Обработка заполнения, разработчик может, проанализировав данные заполнения, самостоятельно заполнить реквизиты нового объекта и затем использовать или не использовать возможности стандартной обработки заполнения.

Какие возможны варианты? Мы уже говорили о них, перечислим еще раз.

- Если данные заполнения имеют значение `Неопределено`, это значит, что новый объект создается командой Создать в панели

функций текущего раздела или командой Создать из списка, в котором отсутствуют отборы.

- Если данные заполнения являются ссылкой, то выполняется ввод на основании и разработчик должен самостоятельно обработать это значение: либо сохранить его в реквизите формы, либо на основе этих данных заполнить несколько реквизитов формы.
- Если данные заполнения – это структура, значит, новый объект создается командой Создать из отобранного списка, или в общем случае разработчик создает его программно, передавая эту структуру через методы ОткрытьФорму(), ПолучитьФорму() или Заполнить().

Пример можно посмотреть в демонстрационной базе «Начальное заполнение», модуль документа Накладная, процедура ОбработкаЗаполнения().

В качестве примера рассмотрим процедуру Обработка заполнения документа Накладная (листинг 3.40).

Листинг 3.40. Различные варианты обработки автоматического заполнения

```
Процедура ОбработкаЗаполнения(ДанныеЗаполнения, СтандартнаяОбработка)

    Сообщение = Новый СообщениеПользователю;
    Если ДанныеЗаполнения = Неопределено Тогда
        Сообщение.Текст = "Ввод новой накладной";

    ИначеЕсли ТипЗнч(ДанныеЗаполнения) = Тип("СправочникСсылка.Поставщики") Тогда
        Сообщение.Текст = "Ввод накладной на основании поставщика";
        Поставщик = ДанныеЗаполнения;

    ИначеЕсли ТипЗнч(ДанныеЗаполнения) = Тип("Структура") Тогда
        Сообщение.Текст = "Ввод накладной из отобранного списка";

        СтандартнаяОбработка = Ложь;

        ОтобранныйПоставщик = Неопределено;
        Если ДанныеЗаполнения.Свойство("Поставщик", ОтобранныйПоставщик) Тогда
            Поставщик = ОтобранныйПоставщик;

        КонецЕсли;

    КонецЕсли;

    Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре для наглядности в каждом из возможных вариантов выводится информационное сообщение.

При обычном вводе (`Если ДанныеЗаполнения = Неопределено Тогда`) не выполняется никаких действий, документ заполняется стандартным способом – реквизит Склад заполняется своим значением заполнения.

При вводе на основании поставщика (`ИначеЕсли ТипЗнч(ДанныеЗаполнения) = Тип("СправочникСсылка.Поставщики") Тогда`) реквизиту Поставщик устанавливается значение того поставщика, на основании которого введена эта накладная. И также используются возможности стандартного заполнения – реквизит Склад заполняется своим значением заполнения.

При вводе из отобранного списка (`ИначеЕсли ТипЗнч(ДанныеЗаполнения) = Тип("Структура") Тогда`) стандартное заполнение не используется (склад не заполняется), а из переданного отбора берется только значение для реквизита Поставщик (другие элементы отбора игнорируются).

Глава 3.11. Проверка заполнения

Механизм проверки заполнения позволяет автоматически проверить, заполнены ли указанные реквизиты объекта. Такая проверка выполняется при интерактивном вводе объекта, перед его записью. Поля реквизитов, для которых должно проверяться заполнение, автоматически выделяются в форме красным подчеркиванием. Если перед записью объекта эти поля окажутся не заполнены, запись не будет выполнена, а в форме будут выданы сообщения об ошибках, привязанные к незаполненным реквизитам (рис. 3.48).

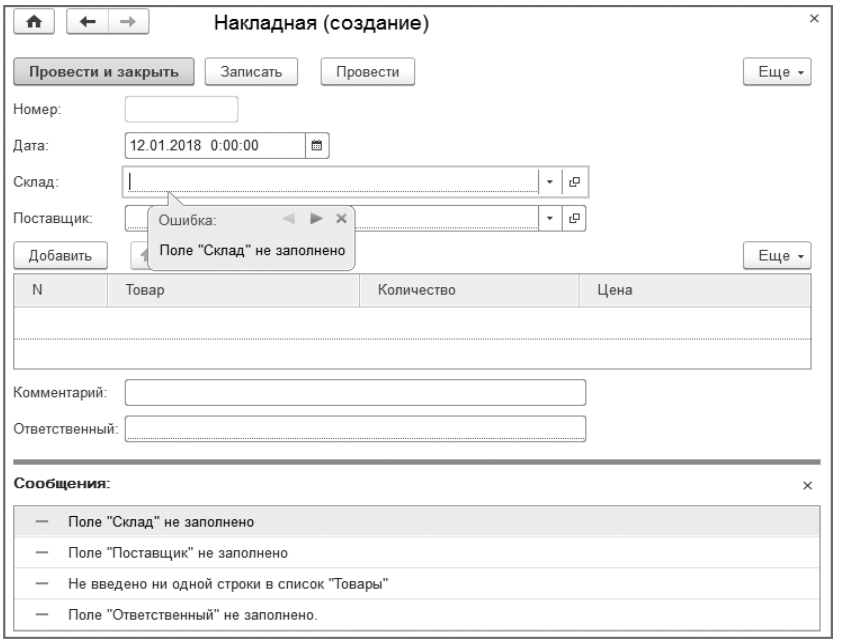


Рис. 3.48. Сообщения пользователю

Механизм проверки заполнения ориентирован исключительно на интерактивную работу, при записи объектов средствами встроенного языка он не вызывается. Но при необходимости имитировать интерактивную запись объекта разработчик может вызвать этот механизм из встроенного языка.

Суть проверки заполнения заключается в том, что у реквизитов объектов конфигурации, у табличных частей, у реквизитов табличных частей и у реквизитов форм есть свойство Проверка заполнения. Стандартное значение этого свойства – Не проверять. Оно означает, что реквизит не участвует в проверке заполнения. Однако если это свойство установить в значение Выдавать ошибку, то перед записью платформа будет проверять, что значение реквизита отличается от значения его типа по умолчанию. А для табличных частей будет проверять, что в табличной части есть хотя бы одна строка (рис. 3.49).

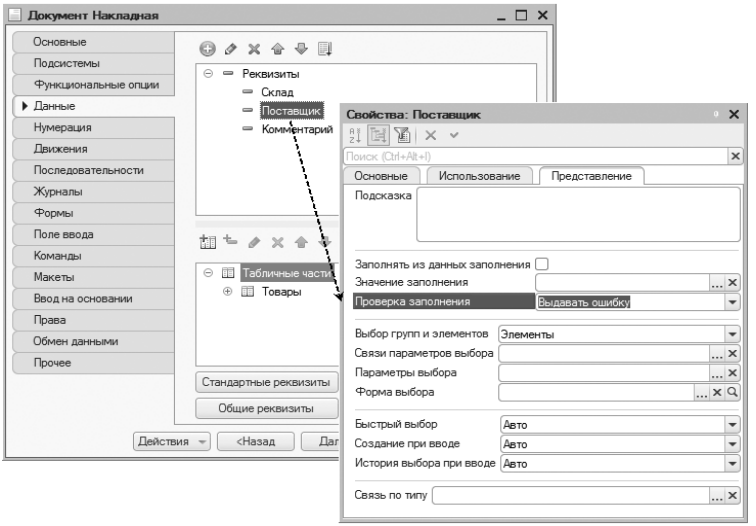


Рис. 3.49. Свойство «Проверка заполнения»

Например, в документе Накладная проверка заполнения может быть указана для его реквизитов, табличной части и реквизитов табличной части. Тогда при попытке провести незаполненный документ платформа выдаст следующие сообщения (рис. 3.50).

А при попытке провести документ с пустой строкой в табличной части сообщения будут следующими (рис. 3.51).

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», документ Накладная, реквизиты Склад, Поставщик, табличная часть Товары, реквизиты табличной части Товар, Количество, Цена, Валюта, форма документа, реквизит Ответственный.

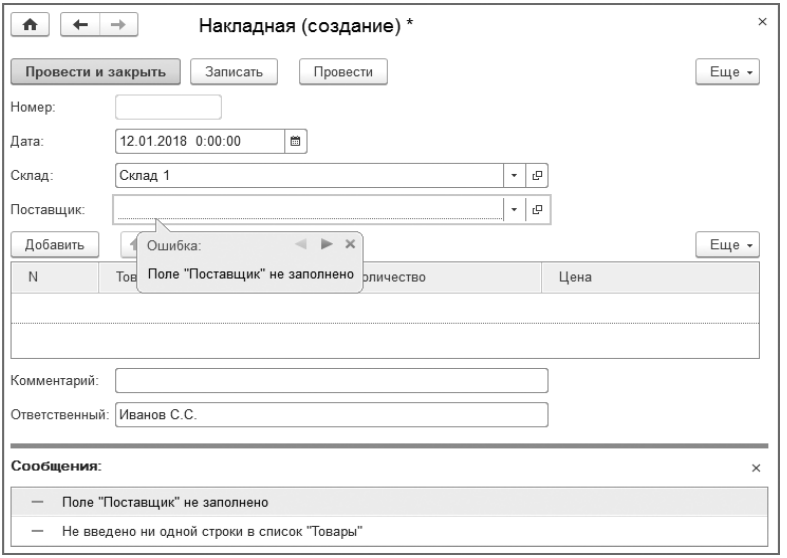


Рис. 3.50. Сообщения о незаполненности реквизитов

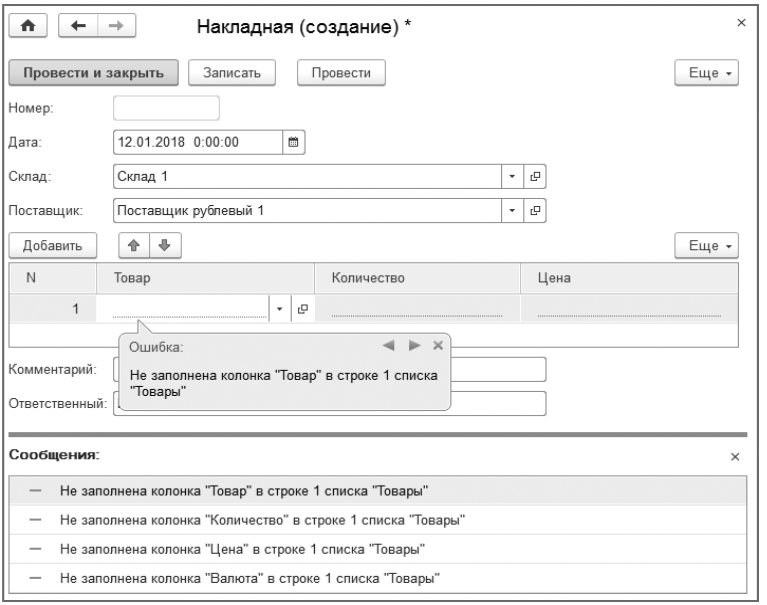


Рис. 3.51. Сообщения о незаполненности табличной части

С помощью мыши или курсора можно перемещаться по сообщениям в окне Сообщения, при этом выделенное сообщение будет показано в форме в специальном окне, размещенном под тем элементом, в котором допущена ошибка. Нажимая кнопки Предыдущее сообщение и Следующее сообщение, пользователь может перемещаться по форме и заполнять пропущенные данные.

Естественно, во встроенном языке разработчик может описать собственные алгоритмы проверки заполнения реквизитов данными. Для этого предназначены два события. Событие формы Обработка проверки заполнения на сервере и событие прикладного объекта, редактируемого в форме, – Обработка проверки заполнения.

Заполнение и проверка заполнения

Прежде чем подробнее рассматривать механизм проверки заполнения, хотелось бы обратить внимание на то, что в платформе есть два интерактивных созвучных между собой механизма: механизм *заполнения* и механизм *проверки заполнения*.

Не следует их путать.

Заполнение – это:

- свойства реквизитов объектов конфигурации Значение заполнения, Заполнять из данных заполнения;
- обработчик объекта Обработка заполнения;
- заполнение выполняется при интерактивном создании нового объекта.

Этот механизм подробно рассматривался в главе 3.10 на стр. 512.

Проверка заполнения – это:

- свойства реквизитов объектов конфигурации Проверка заполнения;
- обработчик формы Обработка проверки заполнения на сервере и обработчик объекта Обработка проверки заполнения;
- проверка заполнения выполняется при интерактивной записи объекта.

Следует учитывать, что у документа проверка заполнения вызывается только при записи с проведением.

Для большей наглядности можно привести следующую схему (рис. 3.52).

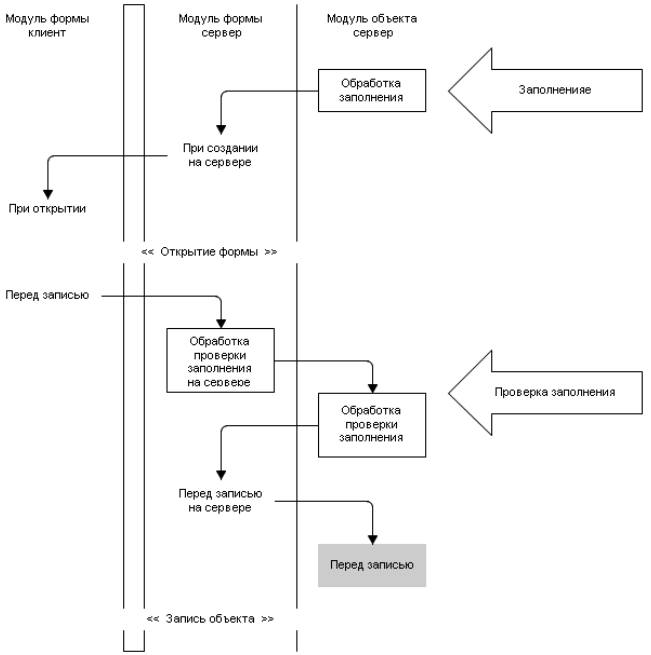


Рис. 3.52. Заполнение и проверка заполнения

Свойство «Проверка заполнения»

Относительно свойства Проверка заполнения хочется сделать следующее замечание. Хотя стандартно для новых реквизитов платформа устанавливает это свойство в значение Не проверять, для некоторых стандартных реквизитов, наоборот, платформа автоматически включает это свойство при создании объектов конфигурации.

Такими реквизитами являются, например, Наименование и Владелец для справочника, Дата для документа и так далее.

Поэтому в тех случаях, когда требуется реализовать нестандартное

Подробнее об этом можно прочитать в документации «1С:Предприятие 8.3.10. Руководство разработчика», раздел 5.6.11.

поведение системы, эти свойства для стандартных реквизитов нужно установить в значение Не проверять или программно исключить эти реквизиты из проверки заполнения (рис. 3.53).

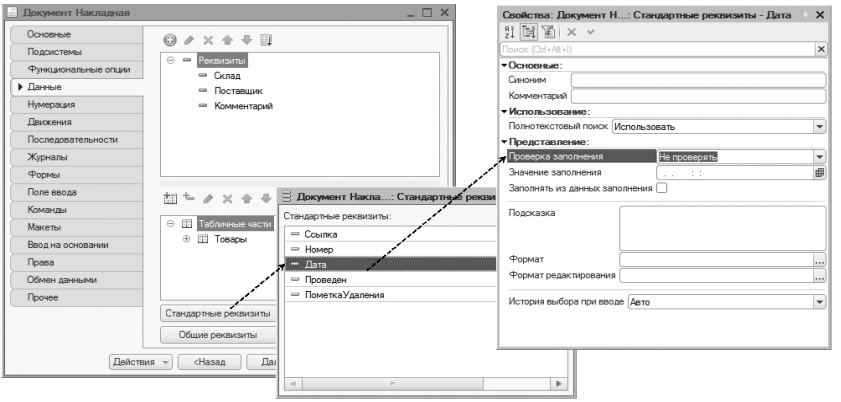


Рис. 3.53. Проверка заполнения для стандартных реквизитов

Программная обработка проверки заполнения

Каким образом разработчик может повлиять на стандартную проверку заполнения, выполняемую платформой?

Для этого у него есть два события. Одно событие – Обработка проверки заполнения на сервере – можно обработать в модуле формы. Другое событие – Обработка проверки заполнения – можно обработать в модуле прикладного объекта.

Все, что относится к проверкам реквизитов основного объекта формы, нужно обрабатывать в модуле объекта.

Обработчик в форме нужен потому, что форма может иметь данные, не относящиеся к объекту, и свои причины для проверки.

И тот и другой обработчик имеют два параметра: `Отказ` и `ПроверяемыеРеквизиты`.

В параметр `ПроверяемыеРеквизиты` платформа передает массив имен тех реквизитов, заполненность которых должна быть проверена.

Разработчик может поступить несколькими способами:

- Проверить заполненность реквизитов самостоятельно и очистить массив ПроверяемыеРеквизиты, чтобы платформа не выполняла их проверку (листинг 3.41).

Листинг 3.41. Самостоятельная проверка всех реквизитов

```
Если Поставщик = Справочники.Поставщики.ПустаяСсылка() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Необходимо заполнить поставщика!";
        Сообщение.Поле = "Поставщик";
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Сообщить();

        Отказ = Истина;

КонецЕсли;

// Проверка остальных реквизитов
// ...

// Очистить массив проверяемых реквизитов, чтобы платформа
// не выполняла их автоматическую проверку.
ПроверяемыеРеквизиты.Очистить();
```

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 1».

- Проверить часть реквизитов самостоятельно, удалить их из массива ПроверяемыеРеквизиты, а оставшиеся реквизиты оставить на проверку платформе (листинг 3.42).

Листинг 3.42. Частичная проверка заполненности реквизитов

```
Если Поставщик = Справочники.Поставщики.ПустаяСсылка() Тогда

        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Необходимо заполнить поставщика!";
        Сообщение.Поле = "Поставщик";
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Сообщить();

        Отказ = Истина;

        // Удалить поставщика из массива проверяемых реквизитов.
        ИндексПоляПоставщик = ПроверяемыеРеквизиты.Найти("Поставщик");
```

```
        Если ИндексПоляПоставщик <> Неопределено Тогда
            ПроверяемыеРеквизиты.Удалить(ИндексПоляПоставщик);
        КонецЕсли;

КонецЕсли;
```

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 2».

- Добавить в массив ПроверяемыеРеквизиты какие-то реквизиты, чтобы платформа проверила и их тоже (листинг 3.43).

Листинг 3.43. Добавление новых реквизитов для автоматической проверки

```
ПроверяемыеРеквизиты.Добавить("Комментарий");
```

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 3».

- Очистить массив ПроверяемыеРеквизиты, чтобы ничего не проверять ни самому, ни платформе (листинг 3.44).

Листинг 3.44. Отказ от автоматической проверки реквизитов

```
ПроверяемыеРеквизиты.Очистить();
```

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 4».

Нужно заметить, что состав массива ПроверяемыеРеквизиты различен в обработчике формы и в обработчике объекта.

В обработчике объекта в этом массиве содержатся имена реквизитов *объекта*, которые требуют проверки (рис. 3.54).

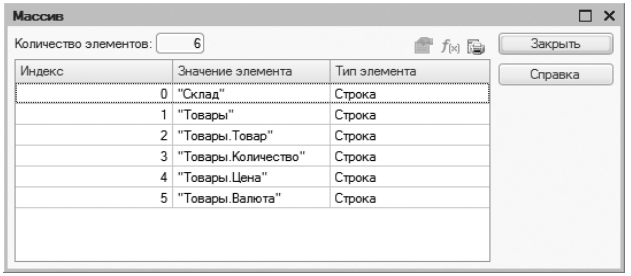


Рис. 3.54. Массив реквизитов, проверяемых в объекте

В обработчике формы содержатся имена реквизитов *формы*, которые требуют проверки. И если есть реквизиты объекта, которые нужно проверять, то содержится имя основного реквизита формы, содержащего данные объекта (рис. 3.55).

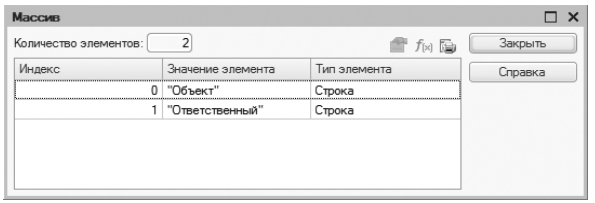


Рис. 3.55. Массив реквизитов, проверяемых в форме

Параметр `Отказ` нужен для того, чтобы отказаться от записи объекта. Не от проверки заполненности, а именно от записи. Чтобы отказаться от проверки объектов, нужно очистить массив `ПроверяемыеРеквизиты`.

Если установить параметр `Отказ` при обработке события в объекте, то после выполнения обработчика Обработка проверки заполнения процесс записи будет прекращен (рис. 3.56).

Если параметр `Отказ` установить в форме, то дальнейшее поведение системы будет зависеть от того, есть ли в списке проверяемых реквизитов основной реквизит формы (см. рис. 3.55). Если есть, отработает проверка заполнения в объекте, и на этом все закончится (см. рис. 3.56).

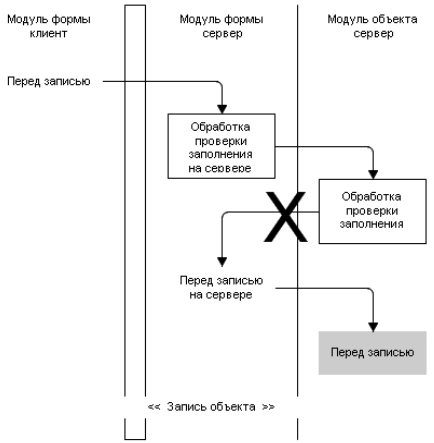


Рис. 3.56. Использование параметра «Отказ» в обработчике объекта

Если нет (например, разработчик очистил этот список), то обработчик в модуле объекта вызываться не будет (рис. 3.57).

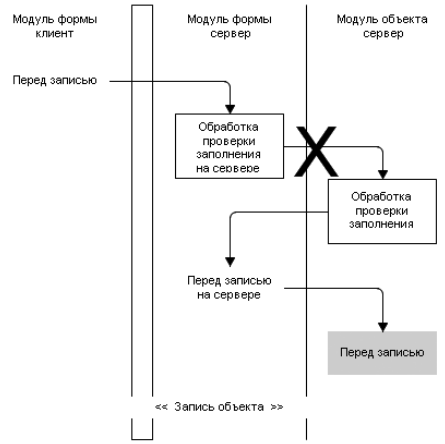


Рис. 3.57. Использование параметра «Отказ» в обработчике формы

Вывод сообщений с привязкой к элементам формы

При самостоятельной проверке заполненности реквизитов, естественно, возникает желание вывести такие же сообщения об ошибках, которые выводит система, чтобы они были привязаны к тем элементам формы, в которых значения не заполнены.

Для этого используется специальный механизм платформы – механизм *сообщений пользователю*. Подробнее об этом механизме можно прочитать в главе 3.12 на стр. 553. Здесь мы рассмотрим только некоторые примеры его использования.

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 5».

Чтобы вывести такое сообщение из модуля объекта (в обработчике события Обработка проверки заполнения), нужно создать новый объект СообщениеПользователю, задать текст этого сообщения.

Также нужно указать, с каким реквизитом формы будет связано это сообщение. Для этого используется свойство Поле (Сообщение.Поле = «Поставщик»). Таким образом мы указываем, что сообщение должно быть привязано к тому элементу формы, который отображает значение реквизита Поставщик.

Мы указали, к какому реквизиту привязать сообщение. Но неизвестно, в какой форме находится этот реквизит. Ведь наш код выполняется в модуле объекта, и у нас нет информации о каких-либо открытых формах.

Сейчас мы не будем подробно углубляться в этот вопрос, скажем лишь, что для того, чтобы сообщение было правильно привязано к форме нашего объекта, нужно задать значения еще двух свойств: ПутьКДанным и КлючДанных.

Эти свойства платформа может заполнить самостоятельно, если выполнить метод УстановитьДанные() и передать в него тот программный объект, в модуле которого мы находимся.

Ну, и собственно для показа сообщения используется метод `Сообщить()`.

Допустим, нужно проверить заполненность реквизита Поставщик (рис. 3.58).

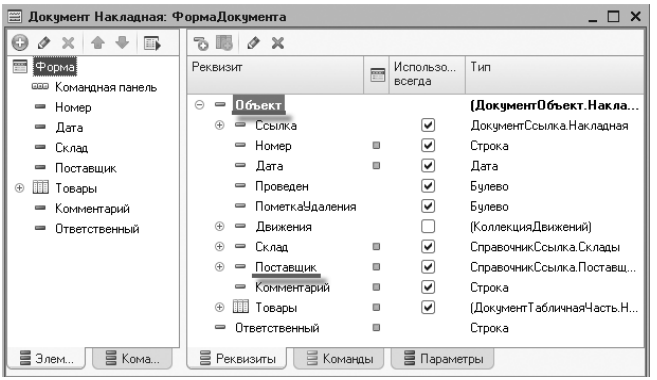


Рис. 3.58. Реквизит «Поставщик», заполненность которого нужно проверить

Это можно выполнить следующим образом (листинг 3.45).

Листинг 3.45. Привязка сообщения к реквизиту объекта

```
Если Поставщик = Справочники.Поставщики.ПустаяСсылка() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Поставщик";  // имя реквизита
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Текст = "Нужно заполнить поставщика!";
        Сообщение.Сообщить();

        Отказ = Истина;  // не выполнять запись

КонецЕсли;
```

В результате, если поставщик не заполнен, мы будем иметь такое сообщение в форме (рис. 3.59).

Обратите внимание, что после выдачи сообщения пользователю мы отменяем дальнейшую запись объекта (`Отказ = Истина`). Если этого не сделать, объект будет записан, несмотря на все наши сообщения, а если выполнялась команда Записать и закрыть, то форма тоже будет закрыта, несмотря на то что в нее были выведены сообщения.

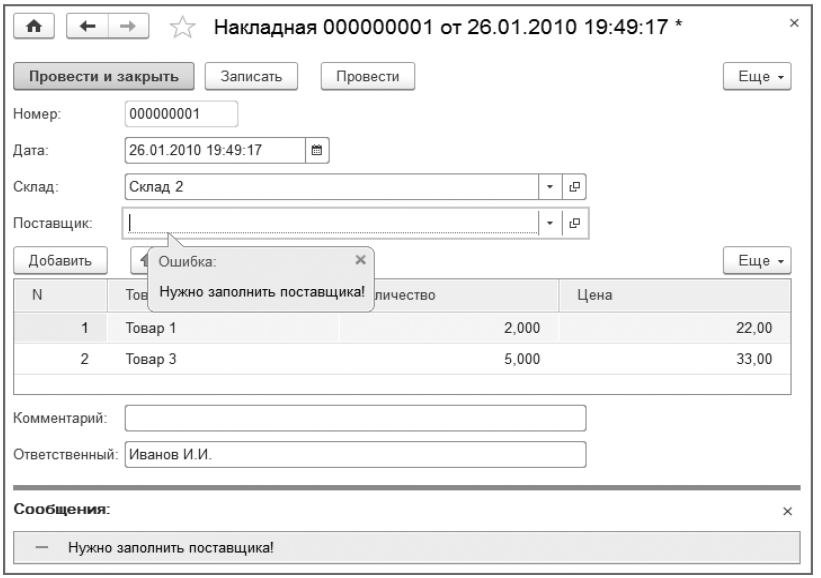


Рис. 3.59. Сообщение о незаполненном реквизите

Теперь рассмотрим ситуацию, когда нужно сообщить об отсутствии строк в табличной части Товары (рис. 3.60).

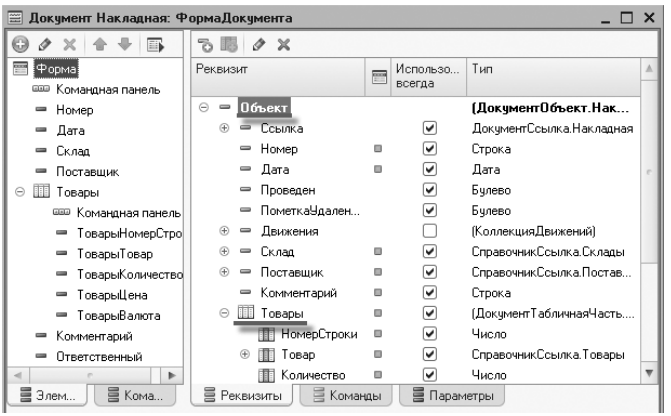


Рис. 3.60. Табличная часть, заполненность которой нужно проверить

Это можно выполнить следующим образом (листинг 3.46).

Листинг 3.46. Привязка сообщения к табличной части

```
Если Товары.Количество() = 0 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Товары";  // имя табличной части
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Текст = "Нужно заполнить список полученных товаров!";
        Сообщение.Сообщить();

        Отказ = Истина;  // не выполнять запись

КонецЕсли;
```

Здесь в свойстве Поле указывается имя табличной части.

В результате, если табличная часть пуста, мы будем иметь такое сообщение в форме (рис. 3.61).

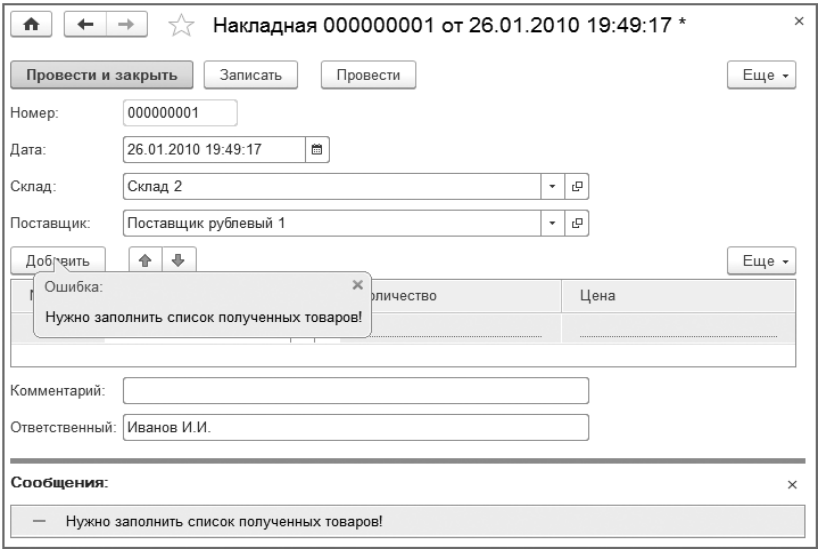


Рис. 3.61. Сообщение о незаполненной табличной части

Теперь, допустим, нужно сообщить о том, что в некоторой строке табличной части не заполнено какое-то поле – например, во второй строке пустое поле Цена (рис. 3.62).

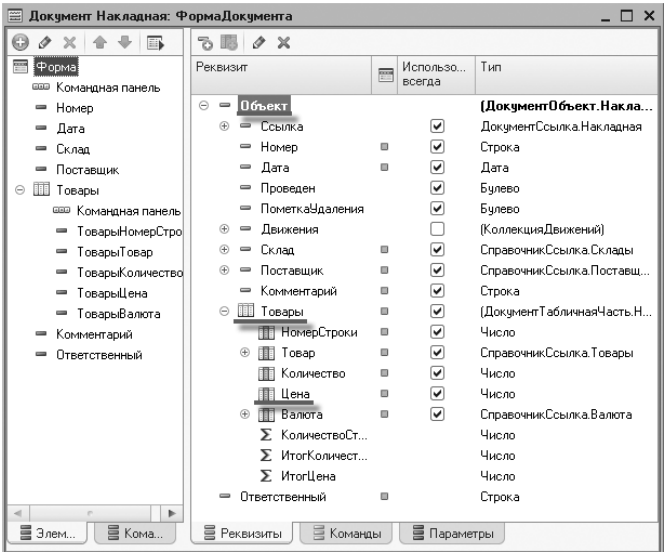


Рис. 3.62. Реквизит табличной части, заполненность которого нужно проверить

Это можно выполнить следующим образом (листинг 3.47).

Листинг 3.47. Привязка сообщения к реквизиту табличной части

```
Если Товары[1].Цена = 0 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Товары[1].Цена";   // индекс строки табличной части и имя поля
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Текст = "Нужно написать цену товара!";
        Сообщение.Сообщить();

        Отказ = Истина;   // не выполнять запись

КонецЕсли;
```

Здесь в свойстве Поле указывается путь к нужному полю табличной части. Сначала с помощью индекса указывается строка (`Товары[1]`), затем через точку указывается поле в этой строке (`Цена`).

В результате, если цена не заполнена, мы будем иметь такое сообщение в форме (рис. 3.63).

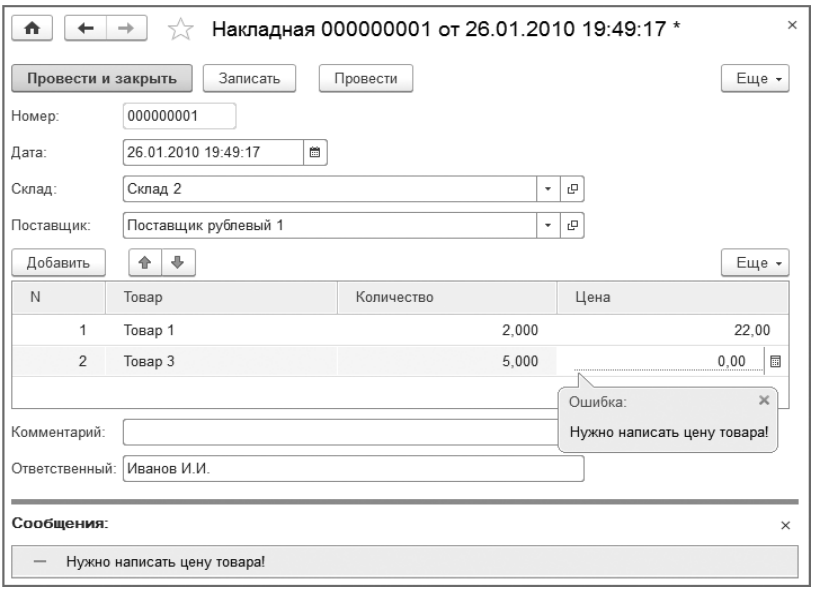


Рис. 3.63. Сообщение о незаполненном реквизите табличной части

В ситуации, когда нужно самостоятельно проверять заполненность каких-либо реквизитов формы (а не реквизитов объекта), все обстоит несколько проще. Такие проверки нужно делать в обработчике события формы Обработка проверки заполнения на сервере.

Поскольку код этого обработчика выполняется в контексте формы, нет необходимости устанавливать для сообщения данные (`Сообщение.УстановитьДанные(…)`). Достаточно лишь указать имя реквизита формы, к данным которого будет привязано сообщение.

Например, нужно проверить реквизит формы Ответственный (рис. 3.64).

Это может выглядеть следующим образом (листинг 3.48).

В результате, если ответственный не заполнен, мы будем иметь такое сообщение в форме (рис. 3.65).

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль формы документа Накладная, снять комментарии с процедуры ОбработкаПроверкиЗаполненияНаСервере.

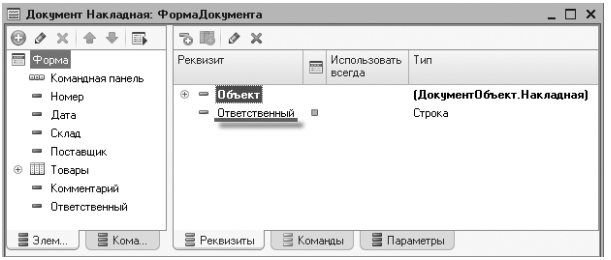


Рис. 3.64. Реквизит формы, заполненность которого нужно проверить

Листинг 3.48. Привязка сообщения к реквизиту формы

```
Если СокрЛП(Ответственный) = "" Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Ответственный";
        Сообщение.Текст = "Нужно написать фамилию ответственного!";
        Сообщение.Сообщить();

        // Удалить ответственного из массива проверяемых реквизитов,
        // чтобы платформа не проверяла его автоматически еще раз.
        ИндексОтветственного = ПроверяемыеРеквизиты.Найти("Ответственный");
        Если ИндексОтветственного <> Неопределено Тогда
                ПроверяемыеРеквизиты.Удалить(ИндексОтветственного);

        КонецЕсли;

КонецЕсли;
```

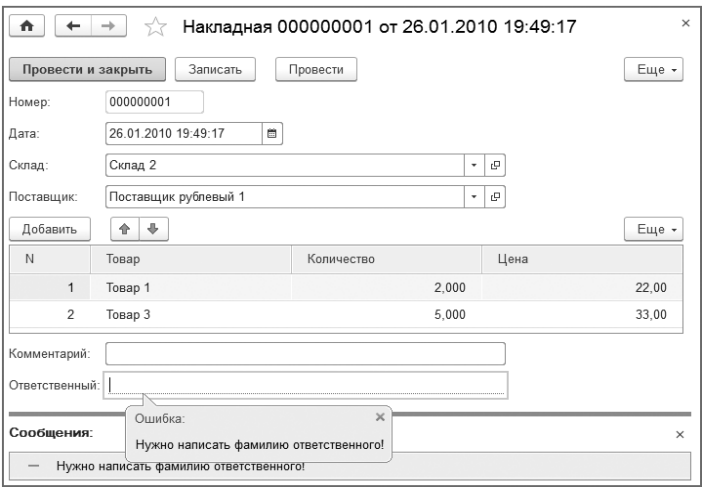


Рис. 3.65. Сообщение о незаполненном реквизите формы

Проверка заполнения и функциональные опции

Использование функциональных опций определяет некоторые особенности проверки заполнения.

Функциональные опции позволяют автоматически скрыть/отобразить элементы интерфейса, которые связаны с функциональностью, не использующейся в данном прикладном решении (или, наоборот, использующейся).

Функциональные опции бывают *независимые* и *параметризуемые*.

Если функциональная опция независимая, то ее значение неизменно для всей конфигурации в целом.

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», функциональная опция УчетПоСкладам.

Оно может храниться, например, в константе типа Булево. Истина – функциональная опция включена, Ложь – выключена (рис. 3.66).

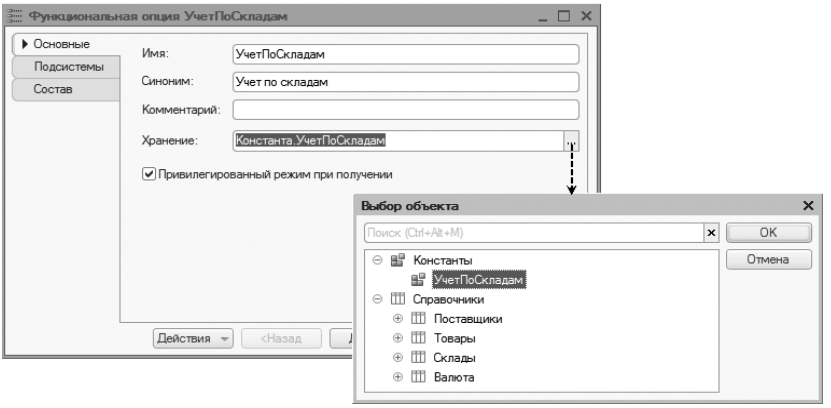


Рис. 3.66. Независимая функциональная опция

В этом случае все просто и понятно.

Если, например, реквизит Склад связан с такой функциональной опцией, то, когда опция выключена, реквизит не отображается в форме и платформа не проверяет его заполненность, не включает его в массив проверяемых реквизитов.

Если функциональная опция включена, реквизит отображается в форме и его заполненность проверяется.

С параметризуемыми функциональными опциями дело обстоит сложнее. Их значение не постоянно для всей конфигурации. Параметризуемая опция в одном документе может иметь значение Истина (быть включенной), а в другом – иметь значение Ложь (быть выключенной). Все зависит от ее параметра. Поясним на примере.

> Пример можно посмотреть в демонстрационной базе «Проверка заполнения», функциональная опция ВалютныйУчет.

Допустим, есть параметризуемая функциональная опция ВалютныйУчет. Она определяет, нужно ли использовать различные валюты при расчетах с поставщиками. Особенность заключается в том, что все зависит от конкретного поставщика. С одним поставщиком расчеты ведутся только в рублях и никак иначе. А с другим поставщиком расчеты могут производиться как в рублях, так и в валюте.

Пусть значение такой функциональной опции хранится в реквизите РасчетыВВалюте справочника Поставщики (рис. 3.67).

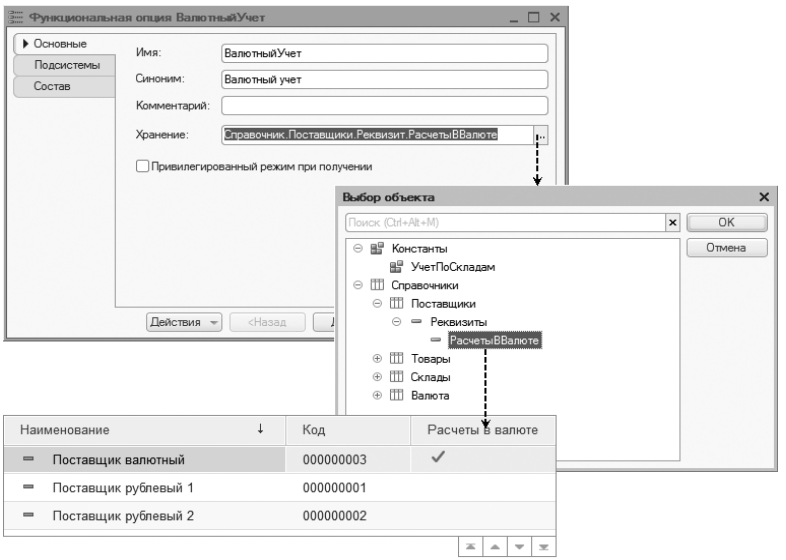


Рис. 3.67. Параметризуемая функциональная опция

Поставщиков в базе данных может существовать много. У одних этот реквизит может иметь значение Истина, у других – Ложь. Какого именно поставщика взять для определения значения функциональной опции?

Для этого как раз и служит параметр функциональной опции (в нашем примере – Поставщик). В каждом конкретном случае он определяет, реквизит какого именно поставщика нужно брать для выяснения значения функциональной опции.

Например, в форме накладной при изменении значения поля Поставщик может выполняться следующий код (листинг 3.49).

Листинг 3.49. Установка параметра функциональной опции

```
ПараметрыОпций = Новый Структура("Поставщик", Объект.Поставщик);
УстановитьПараметрыФункциональныхОпцийФормы(ПараметрыОпций);
```

Таким образом платформе сообщается, что в данный момент параметр функциональной опции должен быть равен тому поставщику, которого выбрал пользователь в поле Поставщик. Имея эту информацию, дальше платформа самостоятельно выясняет значение функциональной опции и скрывает или, наоборот, отображает реквизит табличной части Валюта.

Теперь вернемся к проверке заполнения. Поскольку от разработчика зависит, как и в какой момент в конкретной форме будут установлены параметры функциональных опций, проверяемые реквизиты, связанные с параметризуемыми функциональными опциями, всегда присутствуют в массиве ПроверяемыеРеквизиты. Если в конкретной ситуации заполненность этого реквизита проверять не нужно, разработчик самостоятельно должен удалить его из массива проверяемых реквизитов.

Рассмотрим это на примере.

ПРИМЕЧАНИЕ

Пример можно посмотреть в демонстрационной базе «Проверка заполнения», модуль документа Накладная, процедура ОбработкаПроверкиЗаполнения, снять комментарии для варианта «Пример 6».

Выполним команду Настройки в панели функций из группы команд Сервис (рис. 3.68).

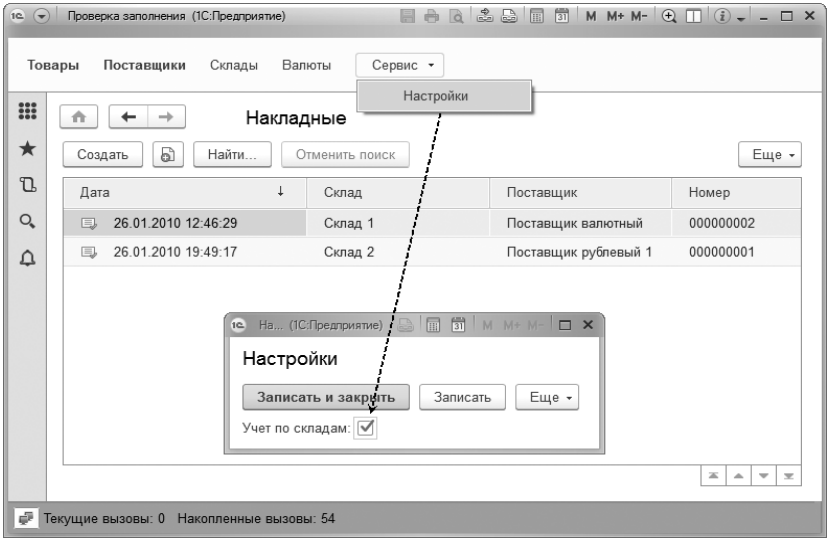


Рис. 3.68. Включение/выключение функциональной опции

Сейчас учет по складам включен, в списке накладных есть колонка Склад, а в самой накладной есть поле Склад. Выключим учет по складам и закроем форму настроек. Создадим новую накладную (рис. 3.69).

В ней уже нет поля Склад.

Заполним все требуемые поля, запишем и закроем накладную.

Никаких сообщений о незаполненности реквизита Склад мы не получим, т. к. платформа автоматически исключила его из числа проверяемых реквизитов, потому что независимая функциональная опция УчетПоСкладам выключена.

Включим учет по складам и снова откроем эту же накладную. В ней появилось поле Склад, и оно не заполнено. Таким его и оставим.

Реквизит Ответственный требует заполнения, но не сохраняется в базе данных. Такова логика работы нашей демонстрационной

формы. Поэтому сейчас для чистоты эксперимента заполним его еще раз.

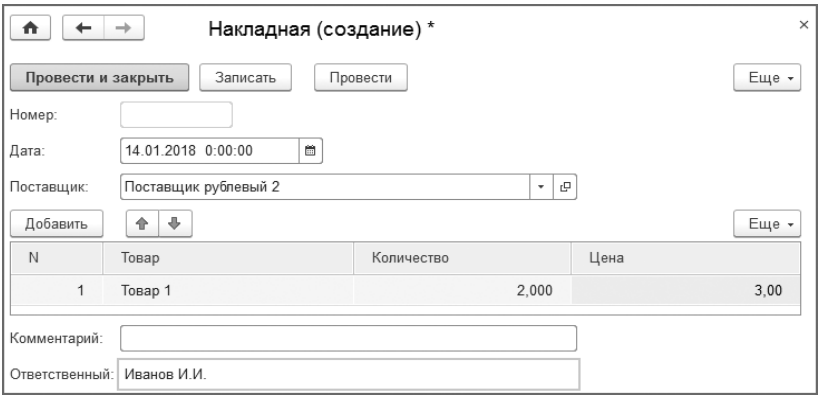


Рис. 3.69. Новая накладная при выключенном учете по складам

После этого попробуем записать документ. Мы получим сообщение о том, что реквизит Склад не заполнен (рис. 3.70).

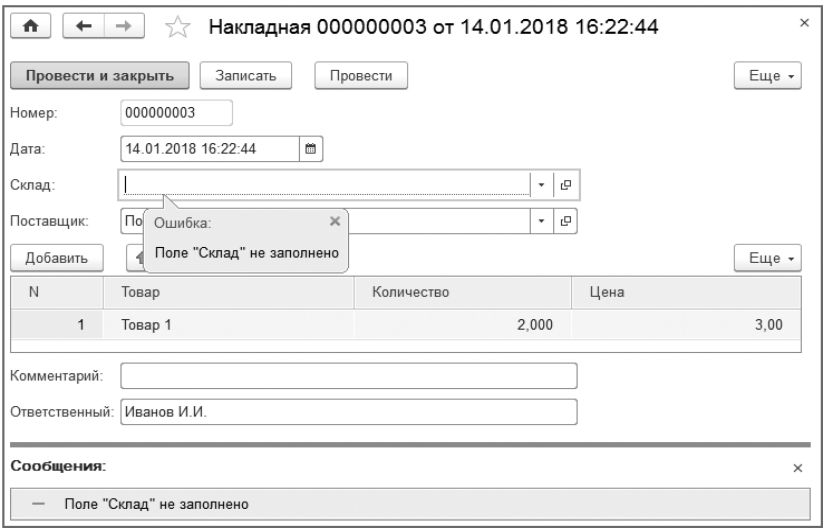


Рис. 3.70. Сообщение о незаполненном поле «Склад»

В данном случае независимая функциональная опция включена, и поэтому платформа автоматически включила связанный с ней реквизит Склад в проверку.

Теперь рассмотрим работу параметризуемой функциональной опции ВалютныйУчет. С этой функциональной опцией связан реквизит табличной части Валюта.

Создадим новую накладную и выберем поставщика Поставщик валютный. В табличной части появится колонка Валюта (рис. 3.71).

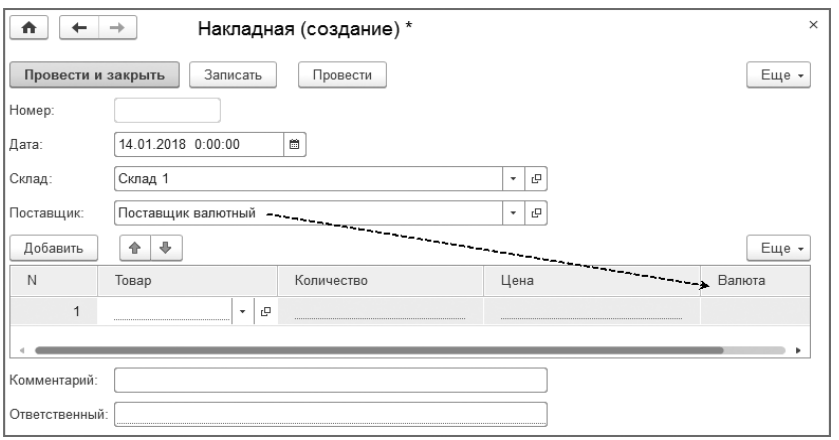


Рис. 3.71. Колонка «Валюта» в табличной части

Заполним все требуемые поля, кроме валюты, и попробуем записать документ. Мы получим сообщение о том, что поле Валюта не заполнено (рис. 3.72).

Теперь, так и не заполняя поле Валюта, выберем другого поставщика – Поставщик рублевый 2. Колонка Валюта исчезнет.

Попробуем записать документ. Документ запишется без сообщений об ошибках.

Таким образом, в одном случае реквизит Валюта проверяется, а в другом – нет. Хотя и в одном, и в другом случае он присутствует в массиве проверяемых реквизитов. Как это достигается?

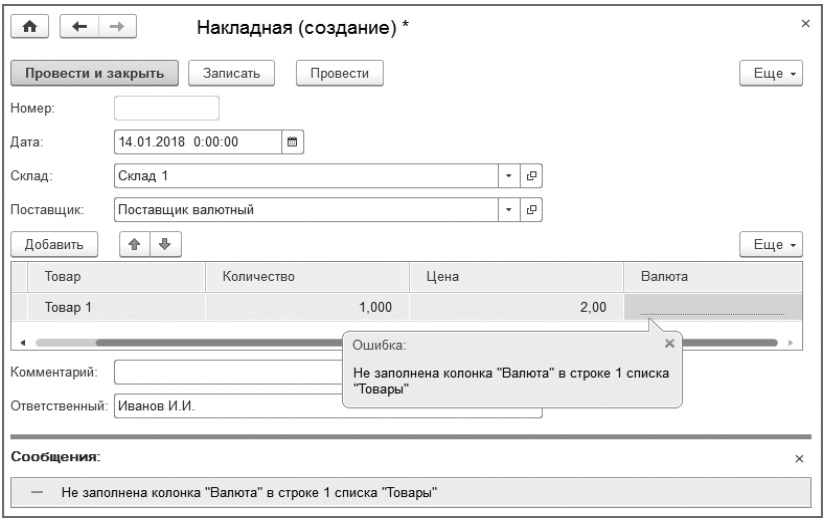


Рис. 3.72. Сообщение о незаполненном поле «Валюта»

Откроем модуль документа, процедуру `ОбработкаПроверкиЗаполнения`, пример 6.

В ней присутствует следующий код, который в зависимости от значения функциональной опции перед записью исключает реквизит Валюта из проверки (листинг 3.50).

Листинг 3.50. Исключение реквизита из проверки в зависимости от значения функциональной опции

```
СтруктураПараметров = Новый Структура("Поставщик", Поставщик);

// Если функциональная опция "Валютный учет" выключена...
Если НЕ ПолучитьФункциональнуюОпцию("ВалютныйУчет", СтруктураПараметров) Тогда

    // ... не проверять заполненность поля "Валюта" табличной части.
    ИндексПоляВалюта = ПроверяемыеРеквизиты.Найти("Товары.Валюта");
    Если ИндексПоляВалюта <> Неопределено Тогда
        ПроверяемыеРеквизиты.Удалить(ИндексПоляВалюта);

    КонецЕсли;

КонецЕсли;
```

Во встроенном языке существует метод глобального контекста ПолучитьФункциональнуюОпцию(). Он позволяет узнать значение параметризуемой функциональной опции. Кроме имени функциональной опции ему нужно передать и значение ее параметра. Значение параметра передается в виде структуры, которая содержит имя параметра и значение параметра.

В нашем случае такой структурой является СтруктураПараметров, где в качестве значения используется поставщик, выбранный в поле Поставщик накладной.

Таким образом, если метод возвращает значение Ложь (выбран поставщик без расчетов в валюте, функциональная опция выключена), выполняется условие Если…, и из массива проверяемых реквизитов удаляется реквизит табличной части Валюта.

Если функциональная опция включена, ничего не происходит, и платформа проверяет значение реквизита Валюта.

Проверка заполнения и проверка при записи

Еще раз хочется обратить внимание на то, что проверка заполнения – это механизм, ориентированный исключительно на интерактивный ввод данных пользователем. Он автоматически вызывается расширениями форм.

Но могут быть ситуации, когда программно реализуется некоторый алгоритм, который должен имитировать для пользователя интерактивный ввод данных. В этом случае проверку заполнения можно вызвать с помощью методов ПроверитьЗаполнение(), реализованных у объектов, наборов записей регистров и формы.

Важно понимать, что проверка заполнения не является частью записи объекта. Она специально выделена отдельно (рис. 3.73).

Таким образом, есть два отдельных вопроса. Первый вопрос: все ли реквизиты объекта заполнены? Второй вопрос: можно ли записывать объект в информационную базу?

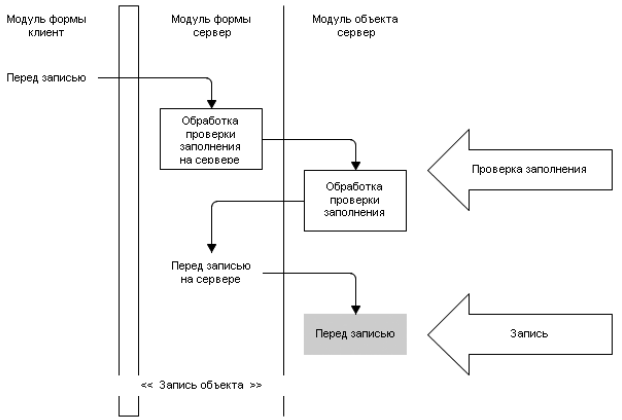


Рис. 3.73. Проверка заполнения и запись объекта

Запись объекта может происходить в самых разных ситуациях: обмен данными, генерация данных обработкой. Это может быть программный код, который «очень хорошо знает», что с объектом нужно сделать и что в нем нужно записать. Поэтому если просто записывать объект из встроенного языка, автоматическая проверка заполнения не вызывается.

Если же запись объекта вызывается из формы, использующей стандартное расширение форм, то это расширение вызовет автоматическую проверку заполнения.

Естественно, в какой-то момент может возникнуть вопрос: что проверять перед записью, а что – в проверке заполнения?

Чтобы решить, нужно, по сути, ответить на вопрос: что является неотъемлемой частью технической логики объекта, не зависящей от способа ввода данных? Такой частью, без которой он существовать не может?

Наверняка какие-то проверки нужно выполнять перед записью. Но они, во-первых, должны быть очень быстрыми (потому что у вас может происходить массовая генерация объектов), во-вторых, должны быть очень «жестокими». То есть вообще в базу данных никогда не должен попасть объект «без такого вот значения».

А проверка заполнения может содержать больший объем проверок, ориентированных именно на то, что не хочется пользователю дать возможность ввести «вот такое» интерактивно.

Пару слов следует сказать о свойстве Запрет незаполненных значений, которое существует у измерений регистров. Если это свойство установлено у измерения, то платформа автоматически будет проверять при записи наборов записей, что измерение заполнено (рис. 3.74).

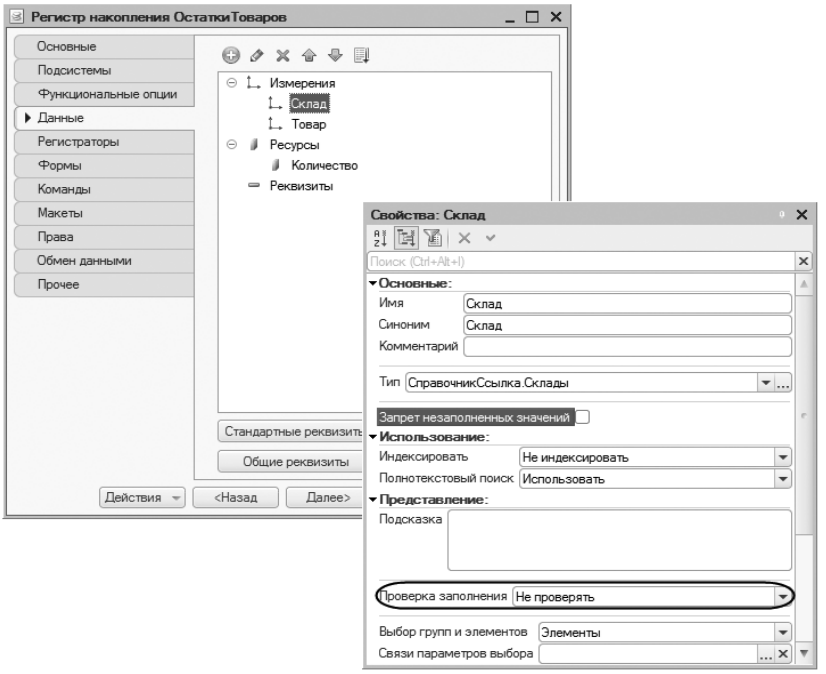


Рис. 3.74. Свойство «Запрет незаполненных значений»

На первый взгляд может показаться, что и там, и там проверяется заполненность, что это одно и то же.

Тут нужно понимать, что проверка заполнения – это исключительно интерактивная проверка; проверка того, что пользователь ввел интерактивно.

В отличие от этого запрет незаполненных значений – это неотъемлемая часть бизнес-логики регистра. Она означает, что записи с незаполненным измерением не имеют смысла в информационной базе, каким бы образом они в нее ни попали: в результате интерактивного ввода или в результате выполнения программного кода.

Глава 3.12. Сообщения пользователю

Сообщения пользователю – удобный инструмент для оповещения пользователя об ошибках в заполнении тех или иных данных. Можно выделить три ключевые возможности сообщений пользователю.

- Во-первых, сообщения выводятся в окно сообщений окна клиентского приложения, в котором открыта форма, и накапливаются в нем. Таким образом, пользователь может последовательно или выборочно выполнять рекомендации, содержащиеся в этих сообщениях.
- Во-вторых, сообщения могут привязываться к элементам формы. В результате пользователь не только быстро находит поле, которое требует исправления, но и быстро выполняет это исправление. Как только сообщение отображается рядом с нужным полем формы, это поле сразу же переходит в режим редактирования.
- В-третьих, сообщения могут относиться не только к той форме, в которой они выведены, но и к другим формам. В этом случае при двойном щелчке на таком сообщении будет открыта форма, к которой относится это сообщение, и в этой форме сообщение будет спозиционировано рядом с нужным элементом формы.

Рассмотрим подробнее, каким образом сообщение определяет тот элемент формы, около которого оно должно быть отображено.

Сообщение имеет три свойства, которые позволяют ему точно позиционироваться возле нужного элемента формы: `КлючДанных`, `ПутьКДанным` и `Поле` (рис. 3.75).

Выполнение любого программного кода всегда начинается на клиенте – из какой-нибудь активной формы или из основного окна.

Все сообщения пользователю платформа выводит в окно формы, идентификатор которой задан в свойстве `ИдентификаторНазначения`. Если идентификатор не указан, то сообщения выводятся в активное окно.

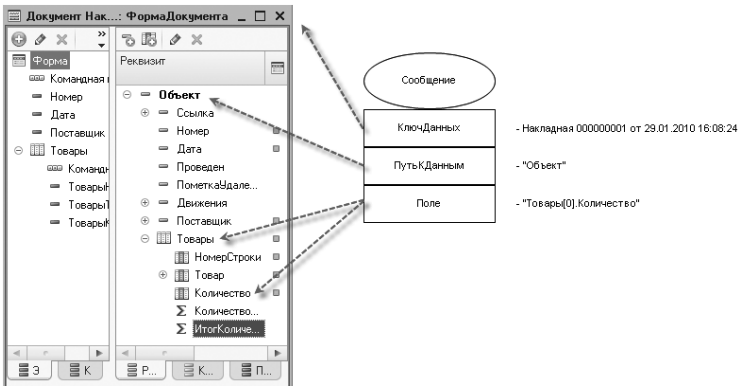


Рис. 3.75. Назначение свойств объекта «СообщениеПользователю»

Когда сообщение оказывается в окне какой-либо формы, первым делом анализируется ссылка, содержащаяся в свойстве КлючДанных.

- Если эта ссылка совпадает с данными, содержащимися в основном реквизите формы, значит, сообщение предназначено этой форме.

  Затем ищется реквизит формы, имя которого указано в свойстве ПутьКДанным.

  После этого ищется подчиненный реквизит, имя которого указано в свойстве Поле.

  В результате после двойного щелчка на сообщении оно отображается рядом с нужным элементом формы.

- Другой сценарий – когда ссылка, содержащаяся в свойстве КлючДанных, не совпадает с данными, содержащимися в основном реквизите формы.

  Это означает, что сообщение предназначено другой форме – основной форме того объекта, ссылка на который содержится в ключе данных.

  В этом случае после двойного щелчка на сообщении будет открыта основная форма объекта, ссылка на который содержится в ключе данных, а далее сообщение будет спозиционировано рядом с нужным элементом формы по принципу, описанному выше.

  Кроме того, выполняется перенос всех сообщений со свойством КлючДанных, равным значению этого свойства текущего сообщения, из окна текущей формы в открытую форму.

Оба рассмотренных случая, когда заполнены все три свойства сообщения, являются наиболее общими и универсальными. Они позволяют правильно воспользоваться сообщением независимо от того, в какой форме оно оказалось.

Но могут быть и другие варианты.

Например, если точно известно, что сообщение может оказаться только в одной-единственной форме (например, когда оно формируется в этой самой форме), можно не указывать КлючДанных. В этом случае будет считаться, что сообщение относится именно к той форме, в которой оно оказалось. При этом если сообщение должно быть привязано не к реквизиту основного объекта, а к реквизиту формы, то также не нужно указывать ПутьКДанным, достаточно указать только Поле.

Есть и еще одна ситуация, которая может возникнуть при работе с сообщениями. Может оказаться так, что для сообщения будут указаны КлючДанных и Поле. А свойство ПутьКДанным не будет заполнено. В этой ситуации сообщение автоматически заполнит свойство ПутьКДанным именем основного реквизита формы. Той, в которой оно оказалось (если ключ данных совпадает), или той, которая будет открыта при двойном щелчке на сообщении.

Сообщения, формируемые при помощи объекта СообщениеПользователю, рекомендуется использовать только для информирования об ошибочных действиях. Если требуется проинформировать пользователя о каком-либо событии, то рекомендуется использовать для этого метод ПоказатьОповещениеПользователя(), о котором рассказывается в разделе «Способы информирования пользователя» на стр. 568.

Теперь рассмотрим все эти ситуации на небольшом практическом примере.

Пример можно посмотреть в демонстрационной базе «Сообщение пользователю».

Допустим, у нас есть обработка ПроведениеДокументов. Задача этой обработки – провести (перепровести) выбранный документ некоторым специальным образом, который не требуется для обычного ежедневного учета. То есть при таком проведении помимо «обычных» данных формируются дополнительные данные, требуемые для расчетов с отдельными поставщиками. Алгоритм расчета этих данных сложен,

таких поставщиков немного, данные требуются эпизодически. Поэтому существует отдельная обработка для получения этих данных.

Наша задача будет заключаться:

- в том, чтобы проверить правильность заполнения всех полей в обработке и выдать сообщения, если какие-то поля не заполнены;
- выполнить проведение документа «сложным» образом, из обработки, и выдать сообщение, если «что-то не так»;
- обеспечить «обычное», ежедневное проведение документа и выдачу сообщений о недостатке списываемых товаров на складе.

Обработка имеет реквизит Документ, в котором содержится ссылка на обрабатываемый документ Накладная. Кроме того, форма обработки имеет реквизит Комментарий, в который должен быть записан произвольный комментарий при интерактивном выполнении обработки (рис. 3.76).

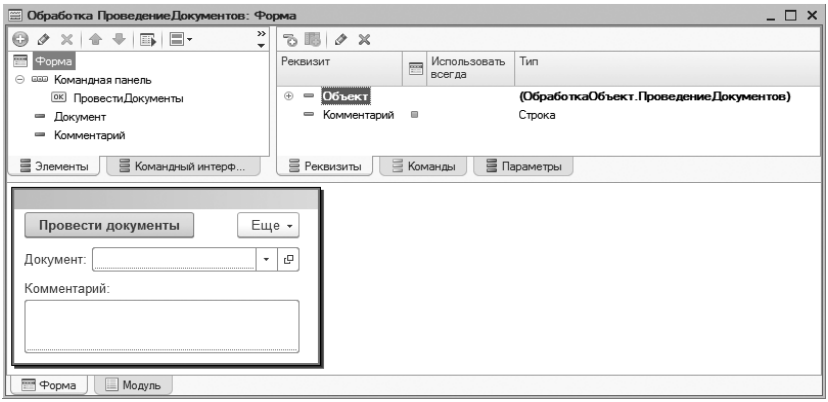


Рис. 3.76. Реквизиты формы

Документ является реквизитом обработки, для того чтобы была возможность программного запуска обработки без использования ее формы.

Для выполнения нашего алгоритма в форме обработки существует локальная команда ПровестиДокументы, которая вызывает процедуру ПровестиДокументы в модуле формы (листинг 3.51).

Листинг 3.51. Обработчик локальной команды формы

```
&НаКлиенте
Процедура ПровестиДокументы(Команда)

    // Использовать стандартный механизм проверки заполнения,
    // реализуемый платформой для обработки.
    Если ПроверитьЗаполнение() Тогда

        // Выполнить собственный алгоритм обработки данных.
        НаСервере();

    КонецЕсли;

КонецПроцедуры
```

В этой процедуре сначала вызывается стандартная проверка заполнения, реализуемая платформой, – `Если ПроверитьЗаполнение() Тогда`.

Метод формы `ПроверитьЗаполнение()` вызывает сначала проверку заполнения в форме, а затем – в объекте обработки. Аналогично тому, как выполняется проверка заполнения при интерактивной записи объектов из формы. Эта тема подробно рассматривалась в главе 3.11 на стр. 525. Сначала будет вызвано событие `Обработка проверки заполнения на сервере` у формы. В обработчике этого события нам нужно проверить заполненность реквизита формы `Комментарий` и вывести сообщение (листинг 3.52).

Листинг 3.52. Обработчик события «Обработка проверки заполнения на сервере»

```
&НаСервере
Процедура ОбработкаПроверкиЗаполненияНаСервере(Отказ, ПроверяемыеРеквизиты)

    ИндексКомментария = ПроверяемыеРеквизиты.Найти("Комментарий");

    Если ИндексКомментария <> Неопределено Тогда
        ПроверяемыеРеквизиты.Удалить(ИндексКомментария);

    КонецЕсли;

    Если СокрЛП(Комментарий) = "" Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Комментарий";
        Сообщение.Текст = "Нужно написать комментарий.";
        Сообщение.Сообщить();

        Отказ = Истина;

    КонецЕсли;

КонецПроцедуры
```

Сначала исключаем комментарий из массива проверяемых реквизитов. Затем, если комментарий «пустой», формируем и выводим сообщение.

В сообщении мы заполняем только свойство Поле. В данном случае активной формой будет форма обработки, и сообщение попадет в нее. Поэтому достаточно указать лишь поле, остальные свойства сообщения не понадобятся (рис. 3.77, 3.78).

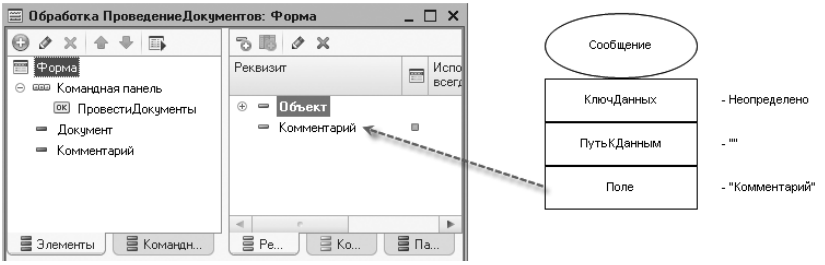


Рис. 3.77. Заполнение свойств сообщения

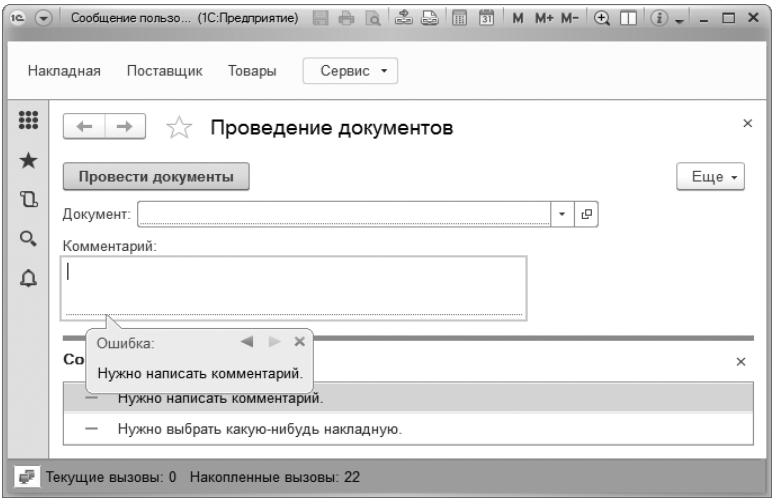


Рис. 3.78. Сообщение, привязанное к реквизиту формы

После того как будет обработано событие в форме, платформа вызовет аналогичное событие у объекта обработки – Обработка

проверки заполнения. В обработчике этого события нам нужно проверить заполненность реквизита обработки Документ и вывести сообщение (листинг 3.53).

Листинг 3.53. Обработчик события «Обработка проверки заполнения»

```
Процедура ОбработкаПроверкиЗаполнения(Отказ, ПроверяемыеРеквизиты)

    ПроверяемыеРеквизиты.Очистить();

    Если Документ = Документы.Накладная.ПустаяСсылка() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Поле = "Документ";
        Сообщение.УстановитьДанные(ЭтотОбъект);
        Сообщение.Текст = "Нужно выбрать какую-нибудь накладную.";
        Сообщение.Сообщить();

        Отказ = Истина;

    КонецЕсли;

КонецПроцедуры
```

Сначала мы очищаем массив проверяемых реквизитов, т. к. все проверки будем выполнять самостоятельно.

Затем проверяем заполненность реквизита Документ и, если он не заполнен (`Если Документ = Документы.Накладная.ПустаяСсылка()`), формируем и выводим сообщение.

Тут все уже несколько сложнее, чем в форме. Проверка заполнения в модуле обработки может быть вызвана и из формы обработки, и программно, без использования формы обработки. А это значит, что формируемое сообщение может попасть не только в форму обработки, для которой оно предназначено, но и в какую-нибудь другую форму.

Поэтому нужно заполнить все свойства сообщения, чтобы, попав в другую форму, сообщение «знало», что нужно открыть форму обработки и именно в ней спозиционироваться возле поля Документ.

Свойство `Поле` мы заполняем самостоятельно: `Сообщение.Поле = «Документ»`. А для того чтобы заполнить два других свойства, `КлючДанных` и `ПутьКДанным`, мы воспользуемся возможностями, предоставляемыми платформой.

Дело в том, что при выполнении некоторых определенных действий платформа запоминает соответствие объекта, который отображается в форме, и имени основного реквизита этой формы. Например, когда в форме объекта выполняется стандартная команда записи или когда в форме вызывается метод `ПроверитьЗаполнение()`.

«Запоминанием» этого соответствия занимается расширение формы, определяемое основным реквизитом. Соответствие запоминается для того, чтобы в модуле объекта можно было бы им воспользоваться для правильного формирования сообщения. Ведь в общем случае, попав в модуль объекта, мы не знаем, из какой формы мы в него попали, не знаем, как в этой форме называется основной реквизит формы. Поэтому, прежде чем перейти в модуль объекта, платформа запоминает, какие данные (ссылку) содержит основной реквизит формы и как он называется.

В случае с обработкой платформа запомнит только имя основного реквизита формы обработки. В платформе не существует типа ссылки на обработку, поэтому запоминать просто нечего.

Затем в модуле обработки мы просто пишем `Сообщение.УстановитьДанные(ЭтотОбъект);` – и у сообщения будет установлено свойство `ПутьКДанным` – «`Объект`». `КлючДанных` останется незаполненным, так как заполнить его просто нечем.

В результате сообщение окажется в форме обработки с незаполненным свойством `КлючДанных`. Поэтому платформа будет пытаться (и это ей удастся) привязать его к форме обработки, используя значения свойств `ПутьКДанным` и `Поле` (рис. 3.79, 3.80).

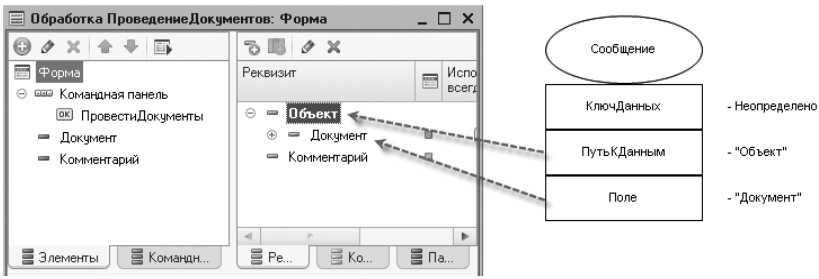


Рис. 3.79. Заполнение свойств сообщения

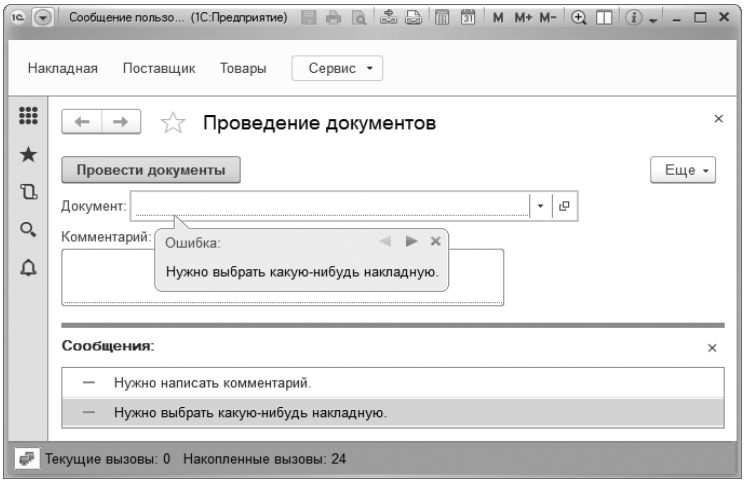


Рис. 3.80. Сообщение, привязанное к реквизиту обработки

Поскольку оба раза при проверке заполнения мы устанавливали параметр `Отказ` обработчиков в значение `Истина`, после обработки события Обработка проверки заполнения дальнейшее выполнение действий останавливается. Метод `ПроверитьЗаполнение()` возвращает `Ложь`, и наша процедура `НаСервере()` не выполняется.

Пока на этом остановимся и обратим внимание на документ `Накладная`.

В модуле документа в процедуре `ОбработкаПроведения` находится код, который формирует движения документа. А после него для примера формируются и выводятся два сообщения.

Предполагается, что наша накладная имеет «два повода» для проведения: обычное повседневное проведение и «специальное» проведение, выполняемое только из этой обработки.

Чтобы различать, каким образом в данный момент проводится накладная, в модуле накладной существует экспортируемая переменная `ПроведениеИзОбработки` (листинг 3.54).

Листинг 3.54. Экспортируемая переменная

```
Перем ПроведениеИзОбработки Экспорт;
```

При любом проведении значение этой переменной сначала устанавливается в Ложь (последняя строка модуля – тело модуля) – листинг 3.55.

Листинг 3.55. Инициализация экспортируемой переменной

```
ПроведениеИзОбработки = Ложь;
```

Это значение будет говорить нам о том, что проведение документа выполняется не из обработки. И в этом случае мы контролируем остатки и выводим сообщение, если товара на складе недостаточно (листинг 3.56).

Листинг 3.56. Сообщение в случае, когда проведение выполняется не из обработки

```
// Проверить, есть ли на складе достаточное количество товара
// в случае оперативного проведения.
// ...
Сообщение = Новый СообщениеПользователю;
Сообщение.Поле = "Товары[0].Количество";
Сообщение.Текст = "На складе есть только 5 единиц товара.";
Сообщение.УстановитьДанные(ЭтотОбъект);
Сообщение.Сообщить();

Отказ = Истина;
```

Здесь, как и в модуле обработки, мы используем возможность платформы самостоятельно устанавливать данные для сообщения (`Сообщение.УстановитьДанные(ЭтотОбъект);`).

Если проведение документа будет выполняться стандартной командой из формы документа, то расширение формы документа автоматически установит соответствие объекта и реквизита формы. Таким образом, КлючДанных и ПутьКДанным будут заполнены, и сообщение будет привязано к нужному полю (рис. 3.81, 3.82).

Если проведение будет выполняться, например, стандартной командой из формы списка (Еще – Провести), то такого соответствия автоматически установлено не будет. Выполнение метода `Сообщение.УстановитьДанные(ЭтотОбъект);` приведет лишь к тому, что будет установлено свойство КлючДанных – ссылка на объект документа. Свойство ПутьКДанным останется незаполненным. При этом сообщение попадет в основное окно приложения (рис. 3.83).

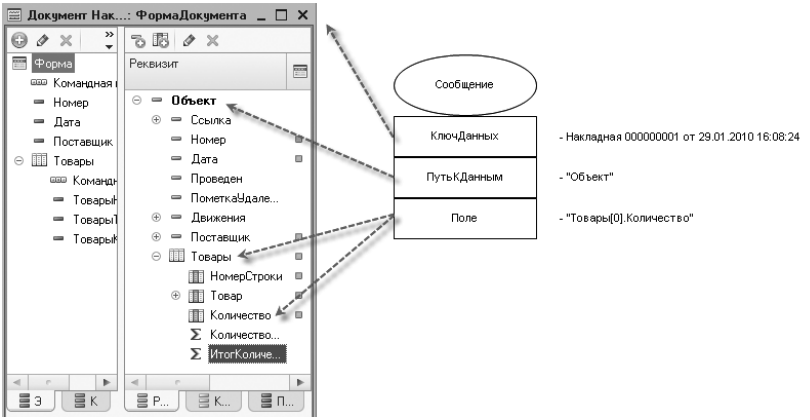


Рис. 3.81. Заполнение свойств сообщения

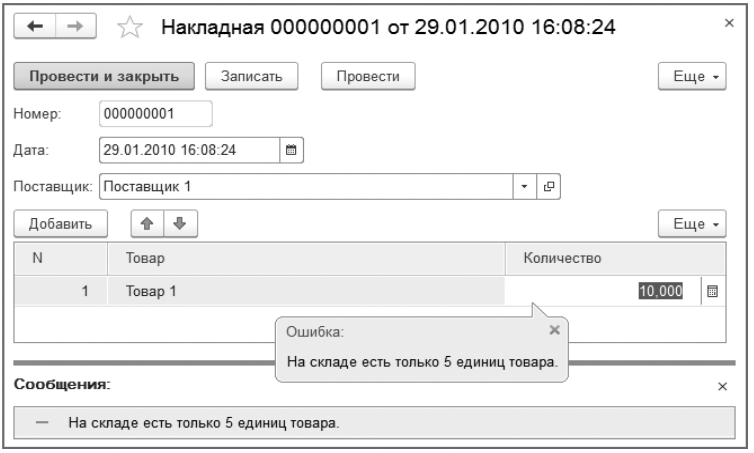


Рис. 3.82. Сообщение, привязанное к реквизиту документа

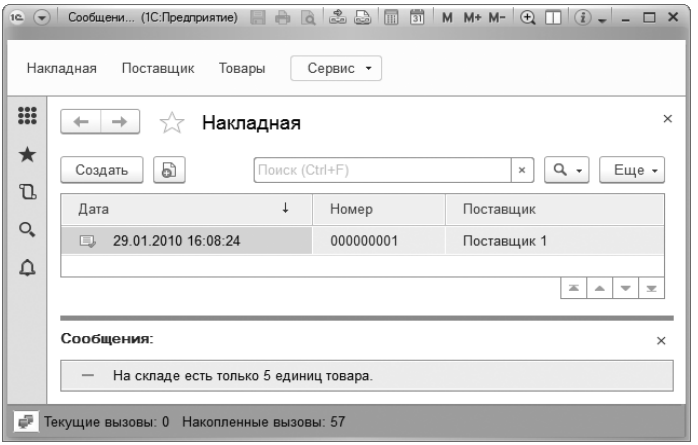


Рис. 3.83. Сообщение в основном окне приложения

При двойном щелчке на этом сообщении будет открыта форма накладной, т. к. у сообщения установлен `КлючДанных`. Платформа автоматически заполнит свойство `ПутьКДанным` именем основного реквизита формы и правильно привяжет его к полю `Количество` (рис. 3.84, 3.85).

Таким образом, если при формировании сообщения мы используем метод `УстановитьДанные()` при любом способе проведения документа, сообщение отработает правильно, в какой бы форме оно ни оказалось.

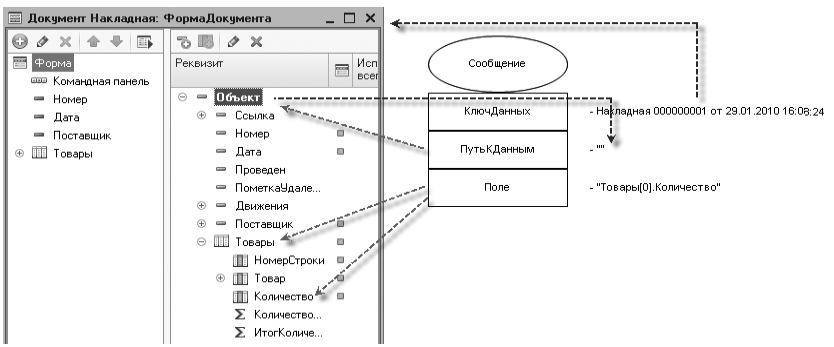


Рис. 3.84. Заполнение свойств сообщения

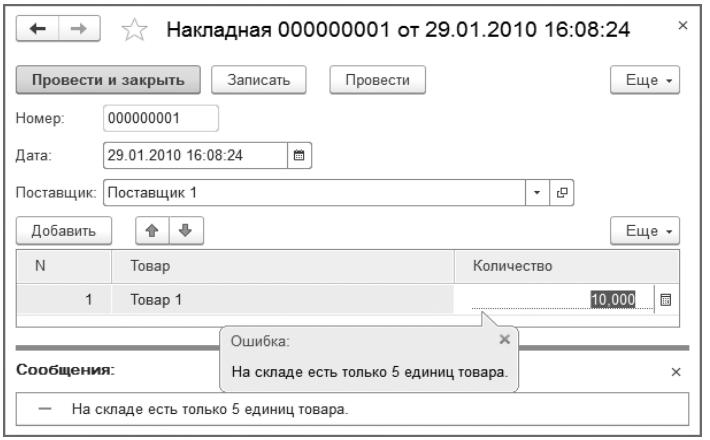


Рис. 3.85. Сообщение, привязанное к реквизиту формы

Теперь вернемся к обработке, заполним поля документа и комментарий и нажмем кнопку Провести документы.

В модуле формы будет вызвана процедура `НаСервере()` (листинг 3.57).

Листинг 3.57. Серверная процедура в модуле формы обработки

```
&НаСервере
Процедура НаСервере()

    // Получить объект документа.
    ОбъектДокумента = Объект.Документ.ПолучитьОбъект();

    // Провести документ.
    ОбъектДокумента.ПроведениеИзОбработки = Истина;
    ОбъектДокумента.Записать(РежимЗаписиДокумента.Проведение);

КонецПроцедуры
```

В ней мы получаем объект документа от ссылки, хранящейся в реквизите Документ.

Затем устанавливаем значение экспортируемой переменной `ПроведениеИзОбработки` модуля документа в `Истина`. В процедуре обработки проведения документа это будет означать для нас, что проведение выполняется «сложным» способом, из обработки.

И после этого вызываем проведение документа в неоперативном режиме (листинг 3.58).

Листинг 3.58. Проведение документа в неоперативном режиме

```
ОбъектДокумента.Записать(РежимЗаписиДокумента.Проведение);
```

В модуле документа, в обработке проведения, анализируем значение переменной ПроведениеИзОбработки, выполняем уже другие проверки и выводим другое сообщение, не такое, как при обычном проведении (листинг 3.59).

Листинг 3.59. Сообщение в случае проведения из обработки

```
Если ПроведениеИзОбработки Тогда

    // Проверить, подходит ли поставщик для выполнения задуманной операции.
    Сообщение = Новый СообщениеПользователю;
    Сообщение.Поле = "Документ";
    Сообщение.ПутьКДанным = "Объект";
    Сообщение.Текст = "Выбран неудачный документ. Расчеты с этим поставщиком
                      были прекращены в прошлом году.";
    Сообщение.Сообщить();
```

Мы хотим, чтобы это сообщение отображалось не в форме документа, а в форме обработки и было привязано к полю Документ.

Поэтому в свойстве Поле мы указываем «Документ», а в свойстве ПутьКДанным – Объект, так как Документ – это подчиненный реквизит объекта обработки. КлючДанных мы не указываем (рис. 3.86).

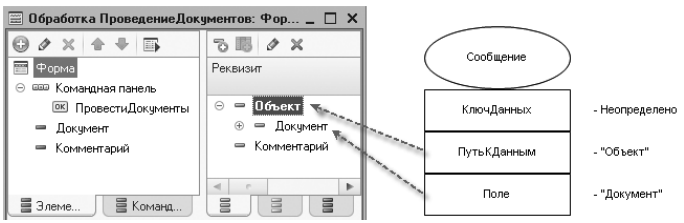


Рис. 3.86. Заполнение свойств сообщения

Во-первых, у нас его нет. Ключом данных должна быть ссылка на объект обработки, а такой тип в платформе отсутствует. Вообще говоря, это не очень хорошо, т. к. снижает универсальность нашего

сообщения. Если оно попадет не в форму обработки, то не сможет правильно отработать.

С другой стороны, вероятность того, что оно попадет не в форму обработки, очень мала. Если предполагается, что обработка будет использоваться только интерактивно, тогда можно допустить, что в другую форму это сообщение не попадет никогда.

А раз так, то свойств `Поле` и `ПутьКДанным` будет достаточно для того, чтобы правильно привязать сообщение в форме обработки. Проверим (рис. 3.87).

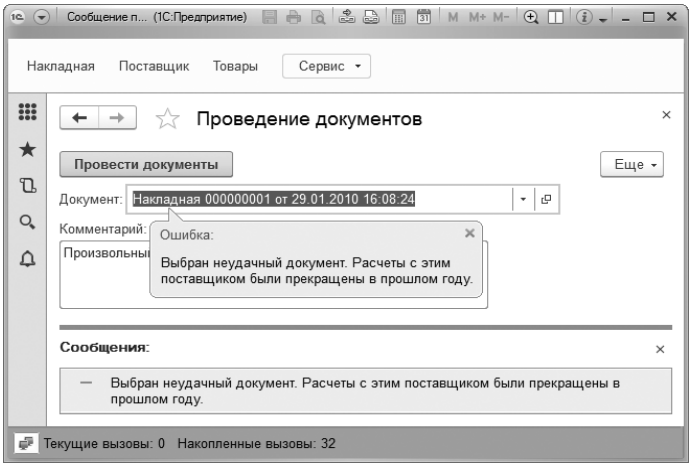


Рис. 3.87. Сообщение, привязанное к реквизиту обработки

При использовании сообщений всегда следует помнить о том, что присутствие сообщений в форме не препятствует ее закрытию.

Поэтому если вывод сообщений используется в стандартных процедурах проверки заполнения, то всегда следует отказываться от продолжения работы. Для этого параметр `Отказ` этих обработчиков нужно устанавливать в значение `Истина`.

В других случаях следует самостоятельно заботиться о том, чтобы форма не была закрыта при появлении в ней сообщений – например, анализируя какой-нибудь служебный реквизит объекта в процедуре формы `ПередЗакрытием()`.

Глава 3.13. Способы информирования пользователя

Начав говорить о сообщениях пользователю, имеет смысл взглянуть более широко на то, какие существуют способы информирования пользователей вообще и в каких ситуациях следует применять тот или иной способ.

Все сообщения, предназначенные пользователю, можно разделить на несколько основных категорий.

- Сообщения о неудачном окончании некоторого процесса. Такие сообщения возникают тогда, когда возникло некоторое исключение. Они информируют пользователя о причинах неудачи. Для таких случаев рекомендуется использовать объект `СообщениеПользователю`. Он был подробно рассмотрен в предыдущей главе (рис. 3.88).

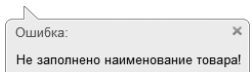


Рис. 3.88. Сообщение пользователю

- Сообщения, которые отражают ход работы. Такие сообщения нужны для информирования пользователя о текущем этапе работы. Для этого хорошо подходит метод глобального контекста `Состояние()`. Он выводит на экран панель состояния, снабженную индикатором (рис. 3.89).

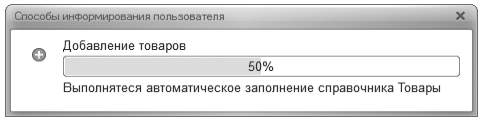


Рис. 3.89. Состояние

  Пример использования этого объекта будет рассмотрен далее.
- Сообщения, с которыми нужно ознакомиться после окончания некоторого процесса, сообщения о проделанной работе. Такие сообщения могут включать информацию о том, что сделано, ее

можно посмотреть, но можно и не смотреть. В случае если факт выполнения команды неочевиден для пользователя, система должна выдавать ту или иную реакцию на любую команду. Неправильно молча «проглатывать» нажатие на кнопку формы. Для таких сообщений следует использовать метод глобального контекста `ПоказатьОповещениеПользователя()`. Оповещение выводится в специальном постепенно затухающем окне в правом нижнем углу экрана (рис. 3.90). Важные для пользователя оповещения (со статусом Важное) запоминаются в центре оповещений, который можно открыть в любое время из панели инструментов.

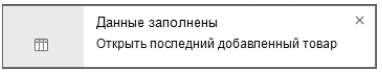


Рис. 3.90. Оповещение пользователя

- Сообщения, с которыми обязательно нужно ознакомиться до начала или после окончания некоторого процесса. Для вывода таких сообщений хорошо подходит метод `ПоказатьПредупреждение()`, выводящий на экран блокирующее окно (рис. 3.91).

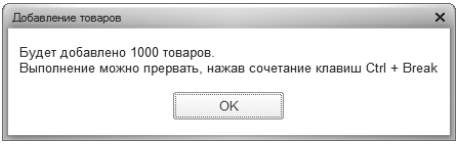


Рис. 3.91. Предупреждение

Если указан тайм-аут, то окно будет закрыто по истечении заданного времени.

Рассмотрим все перечисленные способы на небольшом примере.

> Пример можно посмотреть в демонстрационной базе «Способы информирования пользователя».

Пусть в конфигурации существует справочник Товары, у которого перед записью проверяется заполненность реквизита Наименование. Для этого в модуле объекта, в процедуре `ОбработкаПроверкиЗаполнения`, используется сообщение пользователю (листинг 3.60, рис. 3.92).

Листинг 3.60. Использование сообщения пользователю

```
ПроверяемыеРеквизиты.Очистить();

Если СокрЛП(Наименование) = "" Тогда

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Поле = "Наименование";
    Сообщение.УстановитьДанные(ЭтотОбъект);
    Сообщение.Текст = "Не заполнено наименование товара!";
    Сообщение.Сообщить();

    Отказ = Истина;

КонецЕсли;
```

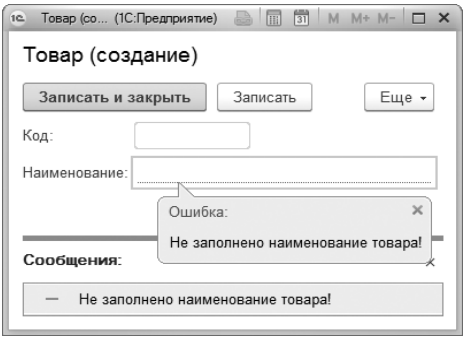


Рис. 3.92. Сообщение пользователю

Кроме того, в конфигурации существует глобальная команда ГенерацияТоваров, которая заполняет справочник Товары элементами в тестовых целях.

Перед началом работы пользователю выдается сообщение о предполагаемых действиях и о том, что он может прервать выполнение этих действий (листинг 3.61).

Листинг 3.61. Использование предупреждения

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

    ПоказатьПредупреждение(
            Новый ОписаниеОповещения("ОбработкаКомандыЗавершение", ЭтотОбъект),
            "Будет добавлено 1000 товаров.
```

```
                |Выполнение можно прервать, нажав сочетание клавиш Ctrl + Break",
                ,
                "Добавление товаров");

КонецПроцедуры
```

Для этого используется предупреждение, которое показывается в блокирующем окне. Это окно не останавливает исполнение программного кода, но блокирует весь интерфейс до тех пор, пока пользователь не ознакомится с этим сообщением и не нажмет кнопку OK (рис. 3.93).

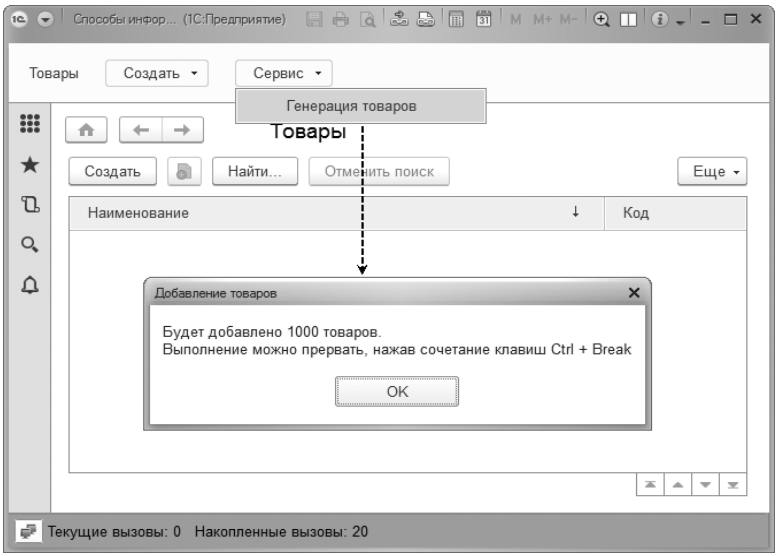


Рис. 3.93. Предупреждение

Предупреждение показывается с помощью немодального метода `ПоказатьПредупреждение()`. Первым параметром в него передается описание оповещения, указывающее на экспортную процедуру обработки оповещения `ОбработкаКомандыЗавершение()`, которая будет выполнена после закрытия окна предупреждения (листинг 3.62).

Таким образом, только после того, как пользователь закроет окно предупреждения кнопкой OK, происходит добавление элементов в справочник Товары.

Листинг 3.62. Обработчик команды «ОбработкаКомандыЗавершение»

```
&НаКлиенте
Процедура ОбработкаКомандыЗавершение(ДополнительныеПараметры) Экспорт

	Для Счетчик = 0 по 9 Цикл

		ПоследняяДобавленнаяСсылка = НаСервере(Счетчик);
		Состояние("Добавление товаров",
		10 + Счетчик * 10,
		"Выполнятся автоматическое заполнение справочника Товары",
		БиблиотекаКартинок.СоздатьЭлементСписка);

		ОбработкаПрерыванияПользователя();

	КонецЦикла;

	// Обновить список товаров на начальной странице
	ОповеститьОбИзменении(ПоследняяДобавленнаяСсылка);

	ПоказатьОповещениеПользователя("Данные заполнены",
	ПолучитьНавигационнуюСсылку(ПоследняяДобавленнаяСсылка),
		"Открыть последний добавленный товар",
		БиблиотекаКартинок.Справочник, СтатусОповещенияПользователя.Важное);

КонецПроцедуры
```

Обратите внимание, что если бы код по добавлению товаров располагался сразу после показа предупреждения, а не в отдельном обработчике оповещения, то товары добавились бы сразу, не ожидая закрытия окна предупреждения. Потому что это окно немодальное и оно не останавливает исполнение программного кода.

Добавление элементов в справочник может быть выполнено только на сервере, и для этого используется серверная функция `НаСервере()`. Она добавляет в справочник 100 элементов (листинг 3.63).

Листинг 3.63. Серверная функция

```
&НаСервере
Функция НаСервере(НомерВызова)

	Для Счетчик = 1 по 100 Цикл
		ОбъектТовара = Справочники.Товары.СоздатьЭлемент();
		ОбъектТовара.Наименование = "Товар_" + Строка(НомерВызова * 100 + Счетчик);
		ОбъектТовара.Записать();

	КонецЦикла;

	Возврат ОбъектТовара.Ссылка;

КонецФункции
```

Для того чтобы пользователь имел возможность прервать процесс добавления элементов в справочник, серверная функция вызывается в цикле несколько раз, то есть добавление элементов выполняется порциями по 100 штук (листинг 3.64).

Листинг 3.64. Вызов серверной функции

```
Для Счетчик = 0 по 9 Цикл

        ПоследняяДобавленнаяСсылка = НаСервере(Счетчик);
        Состояние("Добавление товаров",
                10 + Счетчик * 10,
                "Выполняется автоматическое заполнение справочника Товары",
                БиблиотекаКартинок.СоздатьЭлементСписка);

        ОбработкаПрерыванияПользователя();

КонецЦикла;
```

Каждый раз после выполнения функции проверяется, не нажал ли пользователь комбинацию клавиш Ctrl + Break. Для этого вызывается метод глобального контекста `ОбработкаПрерыванияПользователя()`.

Кроме того, каждый раз после вызова серверной функции в нижнем правом углу экрана выводится окно состояния, отображающее ход процесса с помощью индикатора (`10 + Счетчик * 10`) – листинг 3.65, рис. 3.94.

Листинг 3.65. Вывод состояния

```
Состояние("Добавление товаров",
        10 + Счетчик * 10,
        "Выполняется автоматическое заполнение справочника Товары",
        БиблиотекаКартинок.СоздатьЭлементСписка);
```

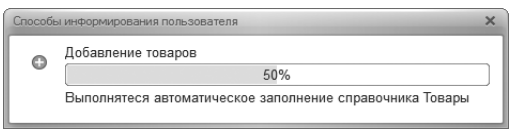


Рис. 3.94. Состояние

В данном случае состояние используется, во-первых, чтобы информировать пользователя о том, что система выполняет какие-то действия,

а во-вторых, чтобы пользователь мог примерно оценить время, оставшееся до окончания действий.

Серверная функция возвращает ссылку на последний элемент, добавленный в справочник Товары.

Эта ссылка используется для того, чтобы после добавления всех элементов обновить список товаров, расположенный на начальной странице (листинг 3.66).

Листинг 3.66. Обновление списка товаров

```
// Обновить список товаров на начальной странице
ОповеститьОбИзменении(ПоследняяДобавленнаяСсылка);
```

Также эта ссылка используется в оповещении, которое выдается по окончании процесса (листинг 3.67).

Листинг 3.67. Вывод оповещения пользователя

```
ПоказатьОповещениеПользователя("Данные заполнены",
        ПолучитьНавигационнуюСсылку(ПоследняяДобавленнаяСсылка),
                "Открыть последний добавленный товар",
                БиблиотекаКартинок.Справочник , СтатусОповещенияПользователя.Важное);
```

Оповещение показывается на несколько секунд в правом нижнем углу экрана. Но, чтобы пользователь позднее мог ознакомиться с ним через центр оповещений, последним параметром в метод `ПоказатьОповещениеПользователя()` передается статус Важное системного перечисления СтатусОповещенияПользователя. Ссылка на последний добавленный товар позволяет открыть его форму сразу же, из оповещения, или позже, из центра оповещений (рис. 3.95).

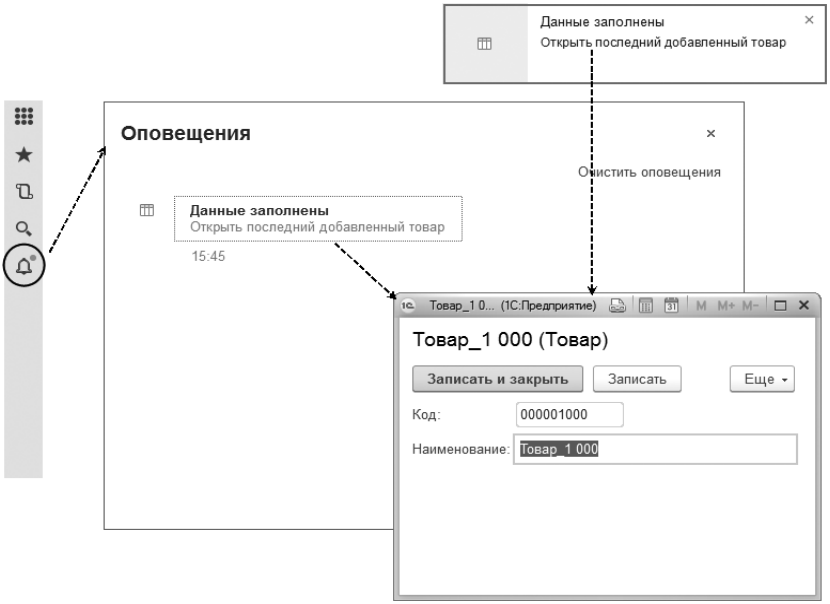


Рис. 3.95. Оповещение во всплывающем окне и в центре оповещений

Сообщение о том, что система закончила выполнение действий, является чисто информационным. В общем случае оно не требует от пользователя никаких действий. Поэтому стандартно используются оповещения со статусом Информация, исчезающие с экрана в случае отсутствия реакции пользователя.

Глава 3.14. Обновление данных в динамических списках

Зачастую запись тех или иных объектов в базу данных выполняется программно, из встроенного языка. Запись в базу данных возможна только на сервере, и в этом случае, естественно, открытые в клиентском приложении формы ничего не знают о том, что данные в базе данных изменились. А ведь эти формы могут содержать списки тех объектов, которые мы изменяли или добавляли в базу данных.

Сама по себе информация в этих списках не обновится. Разработчик должен предпринять специальные действия, для того чтобы динамические списки, существующие в открытых формах, перечитали данные и начали отображать новую, измененную информацию.

Для этого существует несколько способов, которые можно применять в зависимости от конкретной ситуации. Рассмотрим их по порядку, от наиболее универсального к наименее универсальному.

Пример можно посмотреть в демонстрационной базе «Обновление динамических списков», обработка ВариантыОбновления.

Метод «ОповеститьОбИзменении()»

Наиболее простым и наиболее частым является использование метода глобального контекста `ОповеститьОбИзменении()`. В этот метод передается единственный параметр – ссылка на объект (или ключ записи), об изменении которого нужно оповестить формы.

Этот метод уведомит все динамические списки, расположенные в созданных на клиенте формах, об изменении этого объекта, и они обновят свои данные. Но есть особенность: этот метод не обновит те динамические списки, у которых не задана основная таблица.

Преимущество этого способа заключается в том, что нам ничего не нужно знать об открытых формах, не нужно «влезать» внутрь этих форм – платформа все сделает сама.

Пример можно посмотреть в демонстрационной базе «Обновление динамических списков», обработка ВариантыОбновления, локальная команда формы ОповеститьОбИзменении.

Чтобы посмотреть, как будут обновляться различные формы (включая формы, размещенные на начальной странице), в конфигурации созданы «обычная» основная форма списка и специальная форма списка справочника Товары, в которой для динамического списка не назначена основная таблица.

Для удобства эти формы, а также форма обработки ВариантыОбновления помещены на начальную страницу приложения (рис. 3.96).

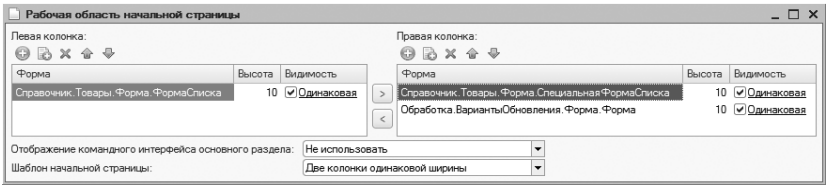


Рис. 3.96. Начальная страница

Кроме того, обе формы списка программно открыты в окнах клиентского приложения из модуля управляемого приложения при начале работы системы (листинг 3.68).

Листинг 3.68. Модуль управляемого приложения

```
Процедура ПриНачалеРаботыСистемы()

    ОткрытьФорму("Справочник.Товары.ФормаСписка");
    ОткрытьФорму("Справочник.Товары.Форма.СпециальнаяФормаСписка");

КонецПроцедуры
```

Предположим, в модуле формы обработки существует серверная функция, добавляющая новый элемент в справочник Товары (листинг 3.69).

Листинг 3.69. Серверная функция

```
&НаСервереБезКонтекста
Функция ДобавитьЭлементНаСервере ()

    ОбъектТовара = Справочники.Товары.СоздатьЭлемент();
    ОбъектТовара.Наименование = "Новый товар " + Строка(ТекущаяДата());
    ОбъектТовара.Записать();

    Возврат ОбъектТовара.Ссылка;

КонецФункции
```

Эта функция возвращает ссылку на тот товар, который был добавлен.

А в процессе выполнения некоторого алгоритма на клиенте выполняется следующий код (листинг 3.70).

Листинг 3.70. Использование метода «ОповеститьОбИзменении()»

```
СсылкаНаНовыйЭлемент = ДобавитьЭлементНаСервере ();
ОповеститьОбИзменении(СсылкаНаНовыйЭлемент);
```

Сначала вызывается серверная функция и добавляется новый товар, а затем полученная ссылка передается в метод `ОповеститьОбИзменении()`.

В результате во всех открытых формах, отображающих список товаров, появится новый товар. Исключением будет лишь специальная форма списка, в которой для динамического списка не назначена основная таблица (рис. 3.97).

В нижней части рисунка для лучшего сравнения результатов обновления формы списка товаров расположены рядом друг с другом в рабочей области основного окна (не на начальной странице) приложения.

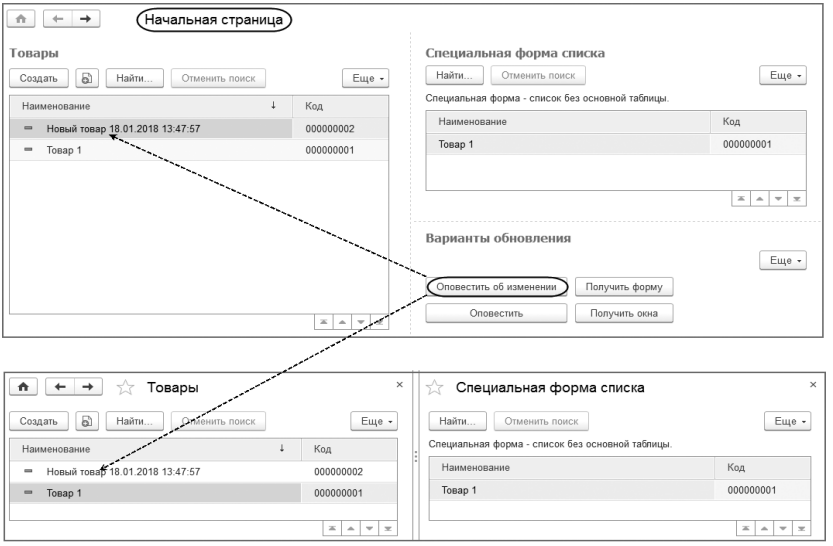


Рис. 3.97. Обновление данных методом «ОповеститьобИзменении()»

Метод «Оповестить()»

Второй способ связан с тем, что мы заранее должны прописать некоторый код в тех формах, в которых нужно что-то обновлять. Но зато тут уже мы можем полностью манипулировать этими формами так, как хочется.

Суть этого способа заключается в том, что существует метод глобального контекста `Оповестить()`. Он отсылает оповещение всем созданным (не обязательно открытым) формам. Если в форме написан обработчик события `ОбработкаОповещения()`, то в этом обработчике можно обработать это сообщение и выполнить нужную модификацию формы.

Пример можно посмотреть в демонстрационной базе «Обновление динамических списков», обработка ВариантыОбновления, локальная команда формы Оповестить.

Например, как и раньше, в серверной процедуре добавляется новый товар. После этого вызывается метод `Оповестить()`, в котором передается идентификатор события – произвольная строка, по которой в форме можно будет понять, какой алгоритм следует выполнить (листинг 3.71).

Листинг 3.71. Использование метода «Оповестить()»

```
СсылкаНаНовыйЭлемент = ДобавитьЭлементНаСервере ();
Оповестить("ОбновитьСписокТоваров");
```

Во всех формах, в которых может понадобиться обновление списка товаров, создается обработчик события `ОбработкаОповещения()`. Например, он может выглядеть следующим образом (листинг 3.72).

Листинг 3.72. Обработчик события «Обработка оповещения»

```
&НаКлиенте
Процедура ОбработкаОповещения(ИмяСобытия, Параметр, Источник)

    Если ИмяСобытия ="ОбновитьСписокТоваров" Тогда
        Элементы.Список.Обновить();

    КонецЕсли;

КонецПроцедуры
```

В результате во всех открытых формах, в которых существует обработчик события `ОбработкаОповещения()`, появится новый товар. В том числе и в тех формах, которые расположены на начальной странице (рис. 3.98).

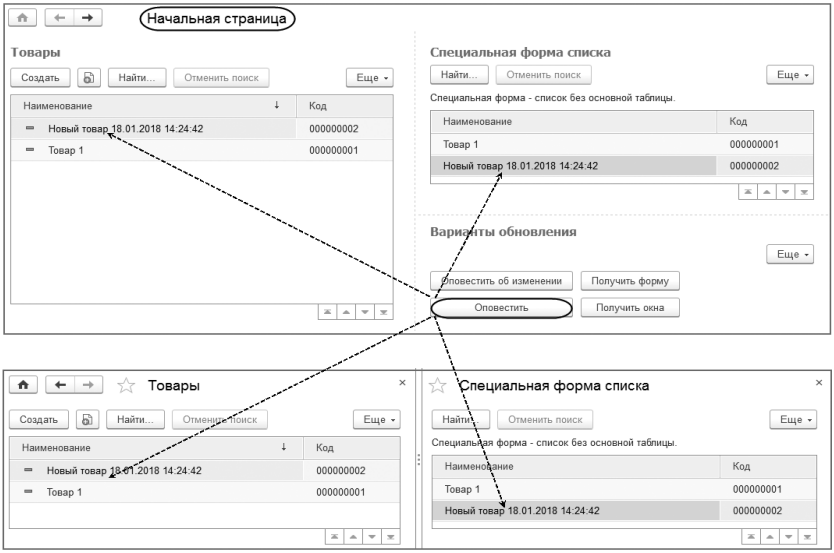


Рис. 3.98. Обновление данных методом «Оповестить()»

Обновление формы извне

Способ, описанный далее, можно использовать тогда, когда нам точно известна форма, в которой нужно что-то обновить, и известно, что эта форма открыта.

Идея заключается в том, чтобы получить саму открытую форму, получить список, расположенный в этой форме, и обновить его.

Для получения формы используется метод глобального контекста `ПолучитьФорму()` (листинг 3.73).

Пример можно посмотреть в демонстрационной базе «Обновление динамических списков», обработка ВариантыОбновления, локальная команда формы ПолучитьФорму.

Листинг 3.73. Получение формы и обновление списка

```
СсылкаНаНовыйЭлемент = ДобавитьЭлементНаСервере ();
Форма = ПолучитьФорму("Справочник.Товары.ФормаСписка");
Форма.Элементы.Список.Обновить();
```

Так как четвертый параметр в этом методе не указывается, то будет получена уже открытая форма. Затем обновляется список, расположенный в этой форме.

В результате в форме списка товаров появится новый товар. Остальные формы останутся без изменений (рис. 3.99).

Рис. 3.99. Обновление данных в конкретном списке

Этот способ имеет существенный недостаток – нужно точно знать устройство формы, как называется ее таблица, отображающая данные динамического списка. Если по каким-то причинам ее имя изменится, форма перестанет обновляться.

Коллекция окон

Последний, четвертый, способ удобен тогда, когда нужно выполнить одинаковые действия со всеми отрытыми формами.

Можно получить коллекцию открытых окон приложения, обойти ее и от каждого окна попробовать получить содержащуюся в нем форму. Затем, как и в предыдущем способе, обновить список, расположенный в этой форме.

Пример можно посмотреть в демонстрационной базе «Обновление динамических списков», обработка ВариантыОбновления, локальная команда формы ПолучитьОкна.

Коллекцию открытых окон приложения можно получить с помощью метода глобального контекста `ПолучитьОкна()` (листинг 3.74).

Листинг 3.74. Работа с коллекцией окон

```
СсылкаНаНовыйЭлемент = ДобавитьЭлементНаСервере();
Окна = ПолучитьОкна();
Для Каждого ТекущееОкно Из Окна Цикл
        Если ТекущееОкно.Содержимое.Количество() Тогда
                Для каждого Форма Из ТекущееОкно.Содержимое Цикл
                        СписокТоваров = Форма.Элементы.Найти("Список");
                        // если есть элемент Список – обновить его
                        Если СписокТоваров <> Неопределено Тогда
                                СписокТоваров.Обновить();
                        КонецЕсли;
                КонецЦикла;
        КонецЕсли;
КонецЦикла
```

В этом примере коллекция открытых окон обходится в цикле. У каждого окна анализируется свойство `Содержимое`, в котором содержится фиксированный массив форм, размещенных на начальной странице, а также открытых в окнах клиентского приложения. Элементы этого массива имеют тип `УправляемаяФорма`. Свойство `Содержимое` для основного окна приложения (с установленным свойством `Основное`) всегда содержит пустой массив.

Если массив `Содержимое` непустой, элементы этого массива обходятся в цикле. В каждой текущей форме ищется элемент с именем `Список`, и если он есть, то он обновляется.

На самом деле алгоритм поиска того, что нужно обновлять, может быть более сложным. Ведь совсем не обязательно таблица будет иметь стандартное имя Список, да и хорошо бы проверить тип объектов, которые отображаются в этом списке… Поэтому здесь просто приведен один из возможных вариантов тех действий, которые требуется выполнить в открытых формах.

В результате список товаров будет обновлен во всех открытых формах, включая формы, расположенные на начальной странице (рис. 3.100).

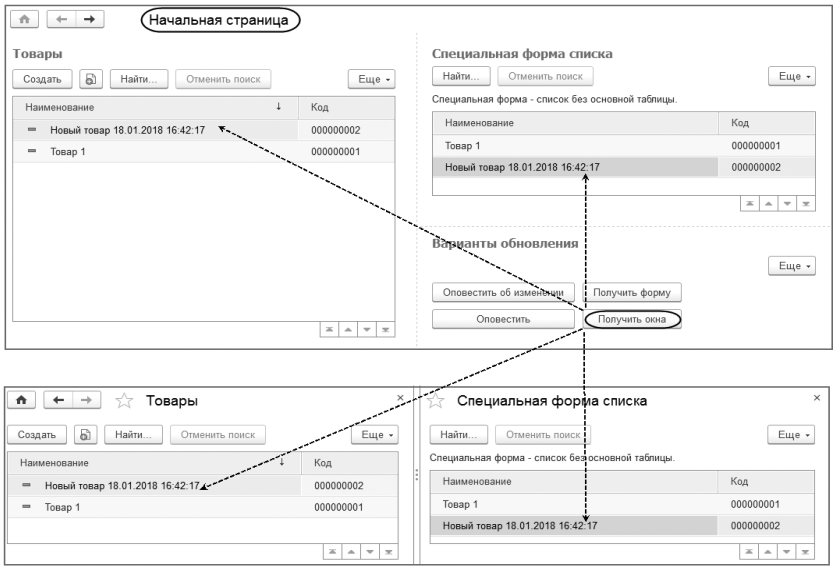


Рис. 3.100. Обновление данных во всех открытых формах

Глава 3.15. Оформление списков

«1С:Предприятие» позволяет различным образом оформлять данные, представленные в списках.

Самые большие возможности оформления предоставляют динамические списки. Для них можно задавать отбор, сортировку, группировать данные по значению и применять условное оформление. Эти возможности доступны разработчику интерактивно в конфигураторе и пользователю, также интерактивно, в режиме 1С:Предприятие.

Возможности по оформлению статических списков (табличных частей, таблиц значений, деревьев значений и пр.) значительно меньше. К ним может быть применено только условное оформление формы, которое разработчик интерактивно задает в конфигураторе.

Важно понимать, что, хотя динамические списки также можно оформить с помощью условного оформления формы, в которой они находятся, делать это не рекомендуется, т. к. это неэффективно.

Однако все упомянутые возможности оформления списков доступны из встроенного языка. Таким образом, разработчик может не только реализовать собственный алгоритм оформления динамического списка, но и предоставить, например, пользователю возможность задать условное оформление для табличной части документа.

Рассмотрим на двух простых примерах, каким образом можно программно оформить «обычные» и динамические списки.

Динамические списки

Для того чтобы познакомиться с возможностями оформления динамического списка, создадим четыре команды, которые в режиме 1С:Предприятие позволят нам задать отбор, порядок, группировку и условное оформление списка документов.

Отбор

Любое оформление динамических списков возможно на клиенте. Поэтому для программной установки отбора в списке накладных мы будем использовать локальную команду формы списка.

Обработчик этой команды выглядит следующим образом (листинг 3.75).

Пример можно посмотреть в демонстрационной базе «Оформление списков», форма списка документа Накладная, команда Отбор. Это локальная команда этой формы.

Листинг 3.75. Установка отбора динамического списка

```
&НаКлиенте
Процедура Отбор(Команда)

	Отбор = Список.КомпоновщикНастроек.Настройки.Отбор.Элементы;

	Если Отбор.Количество() > 0 Тогда
		Сообщение = Новый СообщениеПользователю;
		Сообщение.Текст = "Отбор уже задан. Команда не выполнена.";
		Сообщение.Сообщить();

		Возврат;

	КонецЕсли;

	УсловиеОтбора = Отбор.Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
	УсловиеОтбора.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("НаКонтроле");
	УсловиеОтбора.ВидСравнения = ВидСравненияКомпоновкиДанных.Равно;
	УсловиеОтбора.ПравоеЗначение = Истина;

	Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);

КонецПроцедуры
```

Список – это реквизит формы, содержащий динамический список. Его свойство Отбор может содержать произвольное количество элементов, описывающих условия отбора в этом списке. Эти условия могут быть заданы разработчиком в конфигураторе либо пользователем в режиме 1С:Предприятие или программно из встроенного языка.

Мы будем получать доступ к настройкам списка, заданным в конфигураторе. Они содержатся в коллекции настроек компоновки данных, к которой мы получаем доступ с помощью свойства

динамического списка КомпоновщикНастроек (Список.КомпоновщикНастроек.Настройки).

Прежде чем ставить собственные условия отбора в списке, мы должны каким-то образом обработать ситуацию, когда в списке уже имеются условия отбора.

В реальной жизни возможны различные алгоритмы, но, поскольку у нас всего лишь демонстрационный пример, мы поступим просто. Если для динамического списка уже заданы какие-то условия отбора, мы не будем предпринимать никаких действий и предложим пользователю самостоятельно удалить имеющиеся условия отбора (листинг 3.76).

Листинг 3.76. Анализ имеющихся условий отбора

```
Отбор = Список.КомпоновщикНастроек.Настройки.Отбор.Элементы;

Если Отбор.Количество() > 0 Тогда

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Отбор уже задан. Команда не выполнена.";
    Сообщение.Сообщить();

    Возврат;

КонецЕсли;
```

Если же никаких условий отбора еще не задано, мы зададим одно условие. Для этого в коллекцию элементов отбора добавим новый элемент (листинг 3.77).

Листинг 3.77. Добавление нового условия отбора

```
УсловиеОтбора = Отбор.Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
```

В простом случае отбора может быть перечислено несколько элементов, которые объединяются по условию И. В более сложных случаях отборов несколько элементов могут объединяться в группы, и для каждой группы можно задать собственное условие объединения элементов: И, ИЛИ, НЕ.

Таким образом, в коллекции условий отбора могут находиться как элементы, так и группы. В нашем простом примере мы добавляем

один элемент, но если вы захотите формировать сложные условия, то нужно будет добавлять группы (Тип("ГруппаЭлементовОтбораКомпоновкиДанных")), устанавливать их свойство ТипГруппы и уже в эти группы добавлять отдельные элементы.

Итак, после того как элемент добавлен, нужно задать условие отбора.

У нашей накладной есть булев реквизит НаКонтроле, который проставляется для накладных, требующих «особого внимания» (рис. 3.101).

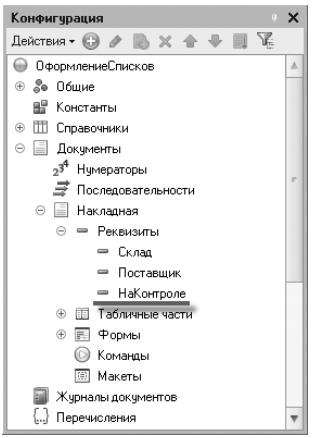


Рис. 3.101. Реквизит «НаКонтроле»

Мы хотим, чтобы в результате наших действий в списке отображались только те накладные, которые на контроле.

Условие отбора задается с помощью свойств ЛевоеЗначение, ВидСравнения и ПравоеЗначение.

В нашем случае левое значение – это поле НаКонтроле (листинг 3.78).

Листинг 3.78. Установка левого значения

```
УсловиеОтбора.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("НаКонтроле");
```

Это поле мы создаем конструктором Новый, указывая имя поля.

Условие сравнения – равенство. Оно задается с помощью системного перечисления (листинг 3.79).

Листинг 3.79. Установка вида сравнения

```
УсловиеОтбора.ВидСравнения = ВидСравненияКомпоновкиДанных.Равно;
```

Правое значение – Истина (листинг 3.80).

Листинг 3.80. Установка правого значения

```
УсловиеОтбора.ПравоеЗначение = Истина;
```

Таким образом должны быть отобраны только те накладные, у которых поле НаКонтроле содержит значение Истина.

И в заключение мы должны загрузить измененные настройки обратно в компоновщик настроек динамического списка (листинг 3.81).

Листинг 3.81. Загрузка настроек в компоновщик настроек

```
Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);
```

Запустим конфигурацию в режиме 1С:Предприятие. Откроется полный список накладных (рис. 3.102).

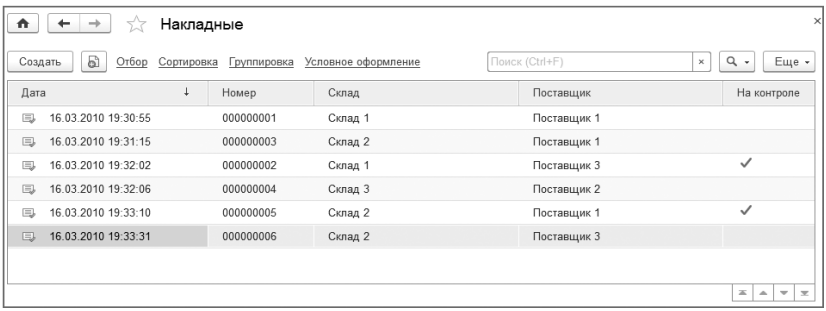


Рис. 3.102. Полный список накладных

После того как мы выполним команду Отбор, в списке останутся только накладные на контроле (рис. 3.103).

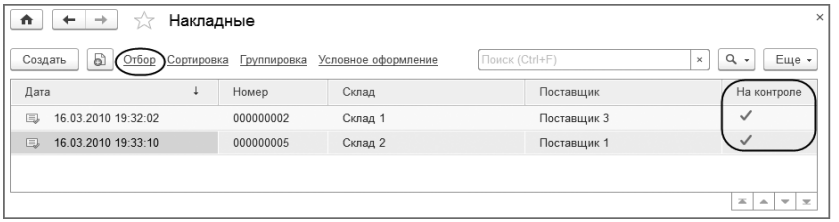


Рис. 3.103. Накладные на контроле

Если мы откроем настройку списка (Еще – Настроить список...), то увидим, что появилось условие отбора (рис. 3.104).

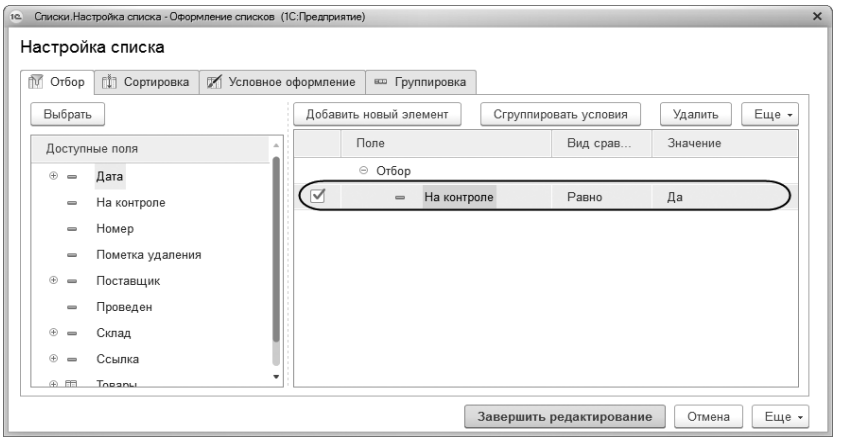


Рис. 3.104. Отбор в настройках динамического списка

Это то самое условие, которое мы добавили из встроенного языка. Пользователь может его изменить, отключить, удалить. Кроме того, у элемента отбора (и у других настроек) существует свойство `РежимОтображения`, с помощью которого он может сделать настройку невидимой и недоступной для изменения в пользовательском режиме (листинг 3.82).

Листинг 3.82. Загрузка настроек в компоновщик настроек

```
УсловиеОтбора.РежимОтображения =
                РежимОтображенияЭлементаНастройкиКомпоновкиДанных.Недоступный;
```

Сортировка

Для сортировки динамического списка будем использовать локальную команду формы со следующим текстом (листинг 3.83).

Листинг 3.83. Установка сортировки динамического списка

```
&НаКлиенте
Процедура Сортировка(Команда)

	Сортировка = Список.КомпоновщикНастроек.Настройки.Порядок.Элементы;

	Если Сортировка.Количество() = 1 Тогда
		Сортировка[0].Использование = Ложь;

	ИначеЕсли Сортировка.Количество() > 1 Тогда
		Сообщение = Новый СообщениеПользователю;
		Сообщение.Текст = "Уже задано несколько условий сортировки. Команда не выполнена.";
		Сообщение.Сообщить();

		Возврат;

	КонецЕсли;

	УсловиеСортировки = Сортировка.Добавить(Тип("ЭлементПорядкаКомпоновкиДанных"));
	УсловиеСортировки.Поле = Новый ПолеКомпоновкиДанных("Склад");
	УсловиеСортировки.ТипУпорядочивания = НаправлениеСортировкиКомпоновкиДанных.Убыв;

	Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);

КонецПроцедуры
```

Как правило, динамический список всегда содержит некоторое условие сортировки. Например, список документов платформа стандартно сортирует по дате документа.

Пример можно посмотреть в демонстрационной базе «Оформление списков», форма списка документа Накладная, команда Сортировка.
Это локальная команда этой формы.

Поэтому анализировать имеющиеся условия сортировки мы будем следующим образом. Если условие одно, будем считать, что это условие задано платформой. Мы будем отключать его использование и добавлять собственное условие. Если условий несколько, сообщим об этом пользователю и не будем выполнять никаких действий (листинг 3.84).

Листинг 3.84. Анализ имеющихся условий сортировки

```
Сортировка = Список.КомпоновщикНастроек.Настройки.Порядок.Элементы;

Если Сортировка.Количество() = 1 Тогда
        Сортировка[0].Использование = Ложь;

ИначеЕсли Сортировка.Количество() > 1 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Уже задано несколько условий сортировки. Команда не выполнена.";
        Сообщение.Сообщить();

        Возврат;

КонецЕсли;
```

После этого добавляем свое новое условие сортировки (листинг 3.85).

Листинг 3.85. Добавление нового условия сортировки

```
УсловиеСортировки = Сортировка.Добавить(Тип("ЭлементПорядкаКомпоновкиДанных"));
```

Поскольку мы собираемся самостоятельно указать, по какому полю сортировать список, мы добавляем «обычный» элемент порядка. Но существует возможность добавить и автоэлемент порядка (`Тип("АвтоЭлементПорядкаКомпоновкиДанных")`). Тогда платформа самостоятельно, на основе имеющегося описания запроса динамического списка, определит, по каким полям нужно выполнять сортировку.

Итак, после того как новый элемент порядка добавлен, нужно указать поле, по которому будет выполняться сортировка, и направление сортировки.

В нашем случае сортировать будем по полю `Склад` (листинг 3.86).

Листинг 3.86. Установка поля, по которому будут выполняться сортировки

```
УсловиеСортировки.Поле = Новый ПолеКомпоновкиДанных("Склад");
```

А направление сортировки будет по убыванию значения этого поля (листинг 3.87).

Листинг 3.87. Установка направления сортировки

```
УсловиеСортировки.ТипУпорядочивания = НаправлениеСортировкиКомпоновкиДанных.Убыв;
```

В заключение загружаем измененные настройки обратно в компоновщик настроек динамического списка (листинг 3.88).

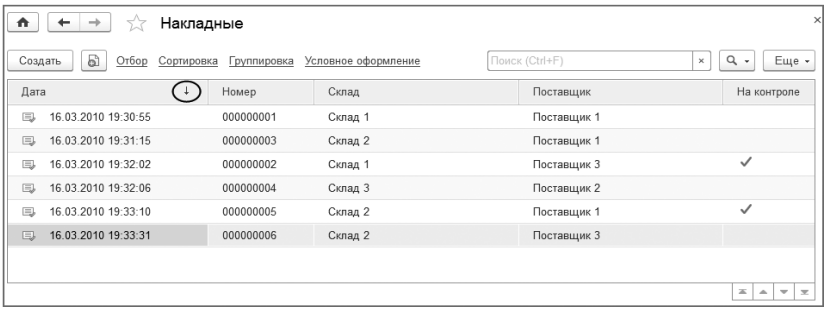
Листинг 3.88. Загрузка настроек в компоновщик настроек

```
Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);
```

Поскольку поле Склад – это ссылочное поле (содержит ссылки на элементы справочника Склады), то сортировка фактически будет выполняться не по самим значениям ссылок, а по их представлениям, то есть по наименованиям складов, которые являются строками.

Запустим конфигурацию в режиме 1С:Предприятие. Откроется список накладных, который стандартно отсортирован платформой по возрастанию даты (рис. 3.105).

На рис. 3.105 показан список накладных без отбора, который мы программно установили в предыдущем примере. Для чистоты эксперимента мы удалили этот отбор из настроек списка в режиме 1С:Предприятие.



Дата	Номер	Склад	Поставщик	На контроле
16.03.2010 19:30:55	000000001	Склад 1	Поставщик 1	
16.03.2010 19:31:15	000000003	Склад 2	Поставщик 1	
16.03.2010 19:32:02	000000002	Склад 1	Поставщик 3	✓
16.03.2010 19:32:06	000000004	Склад 3	Поставщик 2	
16.03.2010 19:33:10	000000005	Склад 2	Поставщик 1	✓
16.03.2010 19:33:31	000000006	Склад 2	Поставщик 3	

Рис. 3.105. Стандартная сортировка списка

После того как мы выполним команду Сортировка, список будет отсортирован по убыванию наименования склада (рис. 3.106).

Если мы откроем настройку списка, то увидим, что существуют два условия сортировки (рис. 3.107).

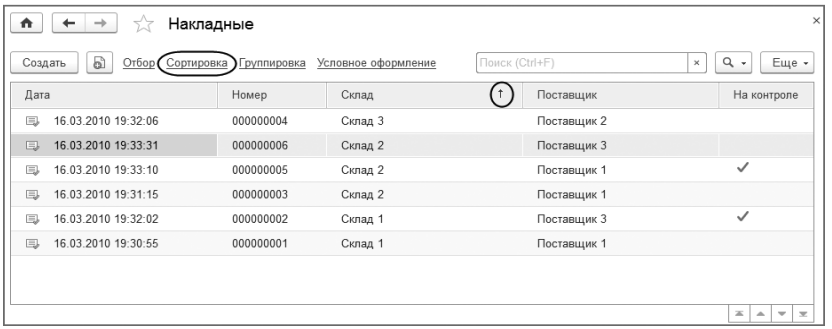


Рис. 3.106. Сортировка по полю «Склад»

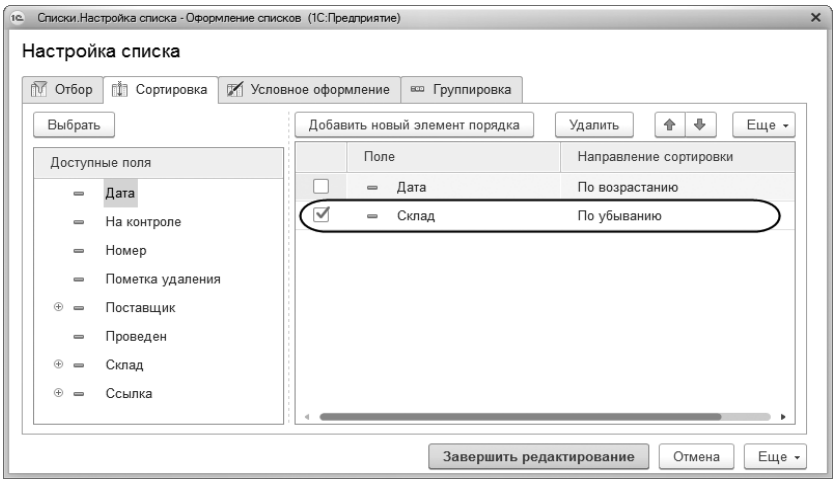


Рис. 3.107. Сортировка в настройках динамического списка

Первое – это стандартная сортировка по дате, которую мы отключили, но не удалили. А второе – это то самое условие, которое мы добавили из встроенного языка. Пользователь может его отключить или удалить.

Группировка

Для группировки динамического списка мы также будем использовать локальную команду формы (листинг 3.89).

Листинг 3.89. Установка группировки

```
&НаКлиенте
Процедура Группировка(Команда)

    СтруктураСписка = Список.КомпоновщикНастроек.Настройки.Структура;

    Если СтруктураСписка.Количество() > 0 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Группировка уже задана. Команда не выполнена.";
        Сообщение.Сообщить();
        Возврат;

    КонецЕсли;

    ГруппировкаСписка = СтруктураСписка.Добавить(Тип("ГруппировкаКомпоновкиДанных"));
    ЭлементГруппировки = ГруппировкаСписка.ПоляГруппировки.
                                    Элементы.Добавить(Тип("ПолеГруппировкиКомпоновкиДанных"));
    ЭлементГруппировки.Поле = Новый ПолеКомпоновкиДанных("Поставщик");
    Элементы.Список.НачальноеОтображениеДерева = НачальноеОтображениеДерева.НеРаскрывать;

    Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);

КонецПроцедуры
```

Здесь, так же как и в случае с отбором, для простоты примера не будем выполнять никаких действий, если условия группировки уже заданы.

Пример можно посмотреть в демонстрационной базе «Оформление списков», форма списка документа Накладная, команда Группировка. Это локальная команда этой формы.

С помощью свойств компоновщика настроек динамического списка (`Список.КомпоновщикНастроек.Настройки.Структура`) мы получаем доступ к коллекции элементов структуры настроек компоновки данных. И проверяем, есть ли элементы в этой коллекции (листинг 3.90).

Листинг 3.90. Анализ имеющихся условий группировки

```
СтруктураСписка = Список.КомпоновщикНастроек.Настройки.Структура;

Если СтруктураСписка.Количество() > 0 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Группировка уже задана. Команда не выполнена.";
        Сообщение.Сообщить();

        Возврат;

КонецЕсли;
```

Если условия группировки не заданы, добавим новый элемент группировки (листинг 3.91).

Листинг 3.91. Добавление нового элемента группировки

```
ГруппировкаСписка = СтруктураСписка.Добавить(Тип("ГруппировкаКомпоновкиДанных"));
ЭлементГруппировки = ГруппировкаСписка.ПоляГруппировки.
                        Элементы.Добавить(Тип("ПолеГруппировкиКомпоновкиДанных"));
```

Сначала мы добавляем в коллекцию элементов структуры настроек компоновки данных элемент типа ГруппировкаКомпоновкиДанных.

Как и в случае с сортировкой, можно использовать автополя группировки, но мы добавим обычное поле, чтобы самостоятельно задать его значение.

В набор полей группировки добавляем элемент типа ПолеГруппировкиКомпоновкиДанных. Группировать список будем по значению поля Поставщик (листинг 3.92).

Листинг 3.92. Добавление поля, по которому будет осуществляться группировка

```
ЭлементГруппировки.Поле = Новый ПолеКомпоновкиДанных("Поставщик");
```

Дополнительно, для «красоты», укажем, что все группы списка должны быть свернуты. Для этого мы обратимся не к динамическому списку (реквизиту), а к таблице формы, которая этот динамический список отображает. Воспользуемся ее свойством НачальноеОтображениеДерева (листинг 3.93).

Листинг 3.93. Установка свойств таблицы

```
Элементы.Список.НачальноеОтображениеДерева = НачальноеОтображениеДерева.НеРаскрывать;
```

У системного перечисления, которое мы использовали, есть еще и два других значения: РаскрыватьВерхнийУровень и РаскрыватьВсеУровни. При необходимости можно использовать их.

Платформа не предоставляет возможности как-либо скрывать/раскрывать отдельные группы сгруппированного динамического списка из встроенного языка. Это можно сделать только интерактивно.

В заключение загружаем измененные настройки обратно в компоновщик настроек динамического списка (листинг 3.94).

Листинг 3.94. Загрузка настроек в компоновщик настроек

```
Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);
```

Запустим конфигурацию в режиме 1С:Предприятие. Откроется список накладных, который стандартно представлен в виде линейного списка (рис. 3.108).

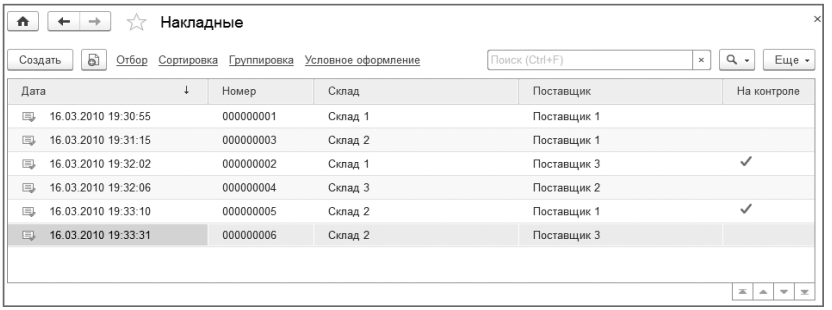


Рис. 3.108. Список без группировки

После того как мы выполним команду Группировка, для каждого поставщика будет создана своя группа, в которой будут находиться документы этого поставщика. Все группы будут свернуты, за исключением той, в которой находится текущая строка списка (рис. 3.109).

На рис. 3.108 показан список накладных без сортировки, которую мы программно установили в предыдущем примере. Для чистоты эксперимента мы удалили эту сортировку из настроек списка в режиме 1С:Предприятие.

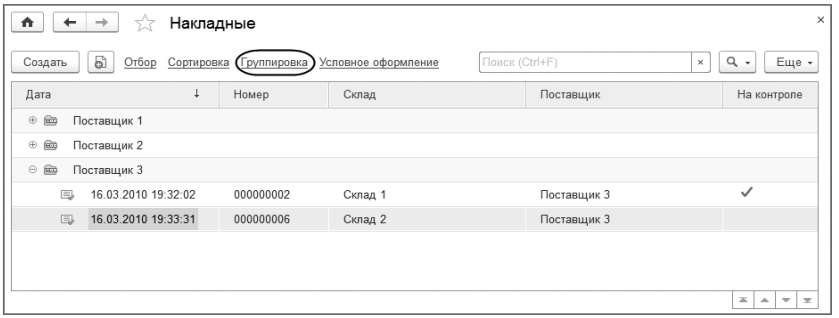


Рис. 3.109. Группировка по поставщику

Если мы откроем настройку списка, то увидим, что появилось условие группировки (рис. 3.110).

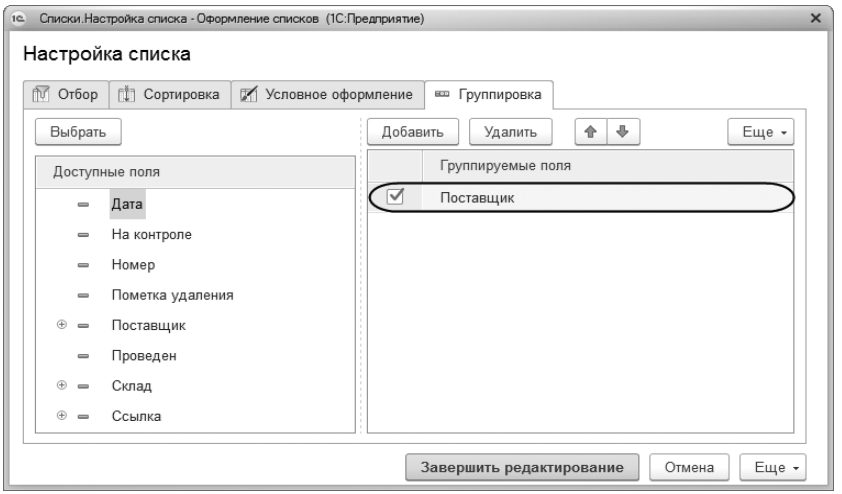


Рис. 3.110. Группировка в настройках динамического списка

Это то самое условие, которое мы добавили из встроенного языка. Пользователь может его изменить, отключить, удалить.

Условное оформление

Для условного оформления динамического списка будем использовать локальную команду формы со следующим текстом (листинг 3.95).

Пример можно посмотреть в демонстрационной базе «Оформление списков», форма списка документа Накладная, команда Условное оформление. Это локальная команда этой формы.

Листинг 3.95. Установка условного оформления

```
&НаКлиенте
Процедура УсловноеОформление(Команда)

	УО = Список.КомпоновщикНастроек.Настройки.УсловноеОформление.Элементы;

	Если УО.Количество() > 0 Тогда
		Сообщение = Новый СообщениеПользователю;
		Сообщение.Текст = "Условное оформление уже задано. Команда не выполнена.";
		Сообщение.Сообщить();

		Возврат;

	КонецЕсли;

	ЭлементУО = УО.Добавить();

	// Оформление: цвет фона – светлый лосось
	ЭлементУО.Оформление.УстановитьЗначениеПараметра("ЦветФона", WebЦвета.ЛососьСветлый);

	// Условие: поле НаКонтроле равно Истина
	ЭлементУсловия = ЭлементУО.Отбор.Элементы.
		Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
	ЭлементУсловия.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("НаКонтроле");
	ЭлементУсловия.ВидСравнения = ВидСравненияКомпоновкиДанных.Равно;
	ЭлементУсловия.ПравоеЗначение = Истина;

	// Оформляемое поле: Номер
	ОформляемоеПоле = ЭлементУО.Поля.Элементы.Добавить();
	ОформляемоеПоле.Поле = Новый ПолеКомпоновкиДанных("Номер");

	Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);

КонецПроцедуры
```

Здесь также для простоты примера не будем выполнять никаких действий, если условное оформление уже задано (листинг 3.96).

Листинг 3.96. Анализ имеющихся элементов условного оформления

```
УО = Список.КомпоновщикНастроек.Настройки.УсловноеОформление.Элементы;

Если УО.Количество() > 0 Тогда

	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Условное оформление уже задано. Команда не выполнена.";
	Сообщение.Сообщить();

	Возврат;

КонецЕсли;
```

Если же условное оформление не задано, добавим новый элемент, описывающий условное оформление (листинг 3.97).

Листинг 3.97. Добавление нового элемента условного оформления

```
ЭлементУО = УО.Добавить();
```

Теперь для этого элемента нужно задать:

- само оформление, которое будет применяться (цвет, жирность шрифта и пр.);
- условие, при выполнении которого это оформление будет применяться;
- поля, которые таким образом будут оформлены.

Оформление задается путем установки значений имеющихся предопределенных параметров оформления. В нашем примере мы указываем, что поля будут выделяться другим цветом фона – «светлый лосось» (листинг 3.98).

Листинг 3.98. Установка оформления, которое будет применяться

```
// Оформление: цвет фона светлый лосось
ЭлементУО.Оформление.УстановитьЗначениеПараметра("ЦветФона", WebЦвета.ЛососьСветлый);
```

Для задания условия, при котором будет применяться оформление, мы используем уже знакомый нам по первому примеру отбор компоновки данных. В коллекцию условий отбора добавим новый элемент и зададим для него левое значение, вид сравнения и правое значение (листинг 3.99).

Листинг 3.99. Добавление условий, при которых будет применяться оформление

```
// Условие: поле НаКонтроле равно Истина.
ЭлементУсловия = ЭлементУО.Отбор.Элементы.Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
ЭлементУсловия.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("НаКонтроле");
ЭлементУсловия.ВидСравнения = ВидСравненияКомпоновкиДанных.Равно;
ЭлементУсловия.ПравоеЗначение = Истина;
```

В результате оформление будет применяться в тех строках, где поле НаКонтроле имеет значение Истина. То есть мы «подсветим» документы, требующие «особого внимания».

В заключение укажем, какие поля будут «подсвечиваться». Если не указывать ничего, то будет «подсвечена» вся строка. Но мы добавим новое поле в коллекцию оформляемых полей и укажем, что оформляться будет только одно поле – Номер (листинг 3.100).

Листинг 3.100. Установка оформляемого поля

```
// Оформляемое поле: Номер.
ОформляемоеПоле = ЭлементУО.Поля.Элементы.Добавить();
ОформляемоеПоле.Поле = Новый ПолеКомпоновкиДанных("Номер");
```

Загружаем измененные настройки обратно в компоновщик настроек динамического списка (листинг 3.101).

Листинг 3.101. Загрузка настроек в компоновщик настроек

```
Список.КомпоновщикНастроек.ЗагрузитьНастройки(Список.КомпоновщикНастроек.Настройки);
```

Запустим конфигурацию в режиме 1С:Предприятие. Откроется список накладных, который стандартно не содержит никакого условного оформления (рис. 3.111).

После того как мы выполним команду Условное оформление, номера документов 000000002 и 000000005 будут выделены цветом фона. Именно эти документы находятся на контроле (рис. 3.112).

На рис. 3.111 показан список накладных без группировки, которую мы программно установили в предыдущем примере. Для чистоты эксперимента мы удалили эту группировку из настроек списка в режиме 1С:Предприятие.

Если мы откроем настройку списка, то увидим, что появилось условное оформление (рис. 3.113).

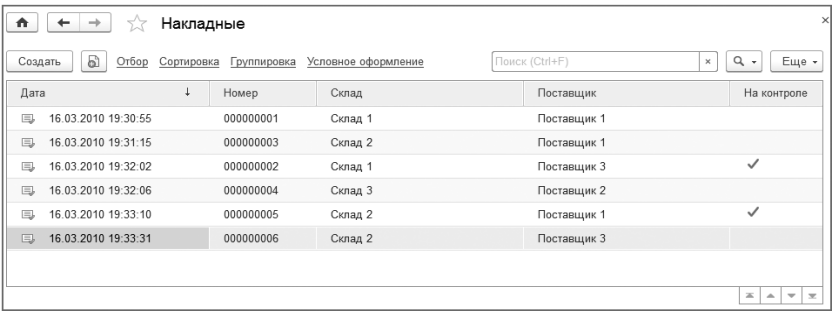


Рис. 3.111. Список без условного оформления

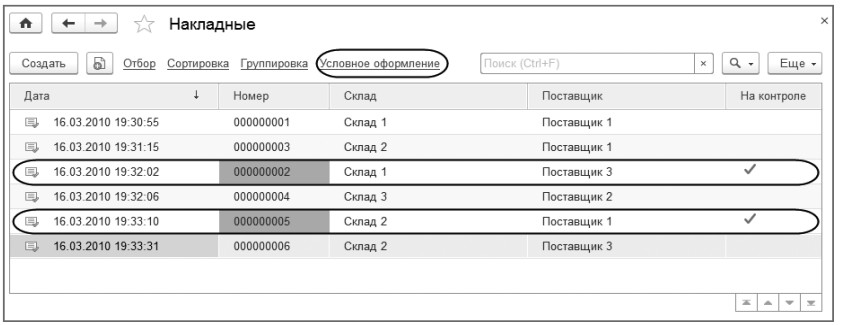


Рис. 3.112. Условное оформление списка

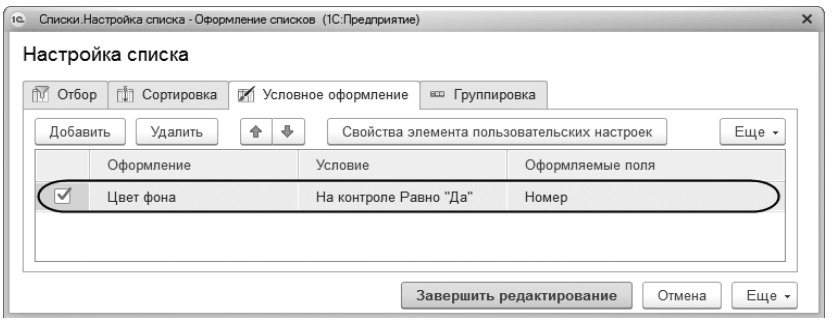


Рис. 3.113. Условное оформление в настройках динамического списка

Это то самое условие, которое мы добавили из встроенного языка. Пользователь может его изменить, отключить, удалить.

Табличная часть

Возможность условного оформления нединамических списков мы рассмотрим на примере табличной части документа. Будем выделять те строки документа, в которых количество товара больше 10.

При оформлении нединамических списков есть несколько особенностей.

Пример можно посмотреть в демонстрационной базе «Оформление списков», форма документа Накладная, команда Условное оформление. Это локальная команда этой формы.

Во-первых, для динамических списков условное оформление является свойством самого реквизита формы. Чтобы оформить нединамические списки, нужно использовать условное оформление самой формы.

Во-вторых, задать оформление динамического списка можно прямо на клиенте. Условное же оформление формы можно задать только на сервере.

В остальном программная работа с условным оформлением нединамических списков ничем не отличается от рассмотренного выше примера.

Для того чтобы оформить табличную часть накладной, создадим локальную команду формы, в которой вызовем контекстную серверную процедуру (листинг 3.102).

Листинг 3.102. Вызов серверной процедуры для оформления формы

```
&НаКлиенте
Процедура УсловноеОформление(Команда)

    ОформитьНаСервере();

КонецПроцедуры
```

А в серверной процедуре выполним условное оформление (листинг 3.103).

Листинг 3.103. Условное оформление формы

```
&НаСервере
Процедура ОформитьНаСервере()

    Если УсловноеОформление.Элементы.Количество() > 0 Тогда
        Сообщение = Новый СообщениеПользователю;
```

```
            Сообщение.Текст = "Условное оформление уже задано. Команда не выполнена";
            Сообщение.Сообщить();

            Возврат;

        КонецЕсли;

        ЭлементУО = УсловноеОформление.Элементы.Добавить();

        // Оформление: цвет фона – зеленая лужайка.
        ЭлементУО.Оформление.УстановитьЗначениеПараметра("ЦветФона", WebЦвета.ЗеленаяЛужайка);

        // Условие: количество в табличной части больше 10.
        ЭлементУсловия = ЭлементУО.Отбор.Элементы.
            Добавить(Тип("ЭлементОтбораКомпоновкиДанных"));
        ЭлементУсловия.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("Объект.Товары.Количество");
        ЭлементУсловия.ВидСравнения = ВидСравненияКомпоновкиДанных.Больше;
        ЭлементУсловия.ПравоеЗначение = 10;

        // Оформляемое поле: товар в табличной части.
        ОформляемоеПоле = ЭлементУО.Поля.Элементы.Добавить();
        ОформляемоеПоле.Поле = Новый ПолеКомпоновкиДанных("ТоварыТовар");

КонецПроцедуры
```

Из особенностей здесь можно упомянуть об именах полей, которые используются в условии оформления и в коллекции оформляемых полей. В динамических списках не вызывает затруднений указать правильное имя поля. Как правило, имена полей в реквизите (динамическом списке) и в таблице совпадают. Так было в предыдущем примере (рис. 3.114).

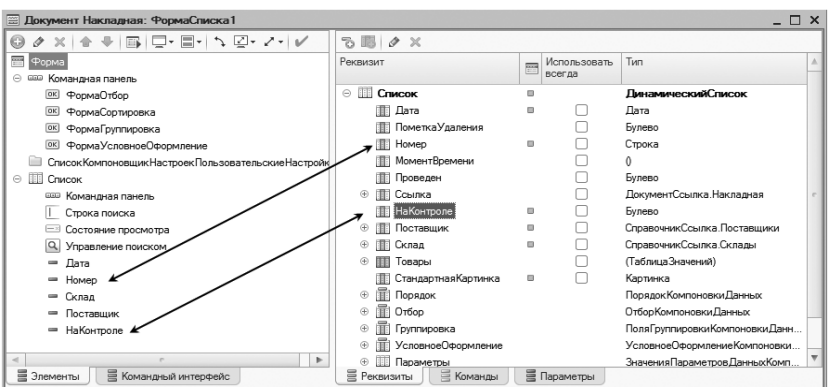


Рис. 3.114. Имена реквизитов и элементов в динамическом списке

Мы накладывали условие на поле реквизита НаКонтроле, а оформляли поле таблицы Номер. При этом можно было не задумываться, в каком случае чье поле указывается.

В случае с табличной частью имена по большей части не совпадают. Поэтому, когда описывается условие оформления, нужно помнить, что указывается поле реквизита формы – Объект.Товары.Количество (листинг 3.104, рис. 3.115).

Листинг 3.104. В условии оформления указывается поле реквизита формы

```
ЭлементУсловия.ЛевоеЗначение = Новый ПолеКомпоновкиДанных("Объект.Товары.Количество");
```

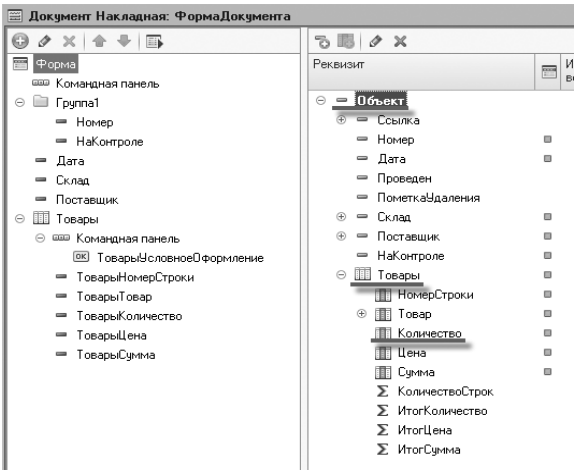


Рис. 3.115. В условии оформления указывается поле реквизита формы

А в коллекции оформляемых полей нужно указывать имя поля таблицы – ТоварыТовар (листинг 3.105, рис. 3.116).

Листинг 3.105. В коллекции оформляемых полей указывается имя поля таблицы

```
ОформляемоеПоле.Поле = Новый ПолеКомпоновкиДанных("ТоварыТовар");
```

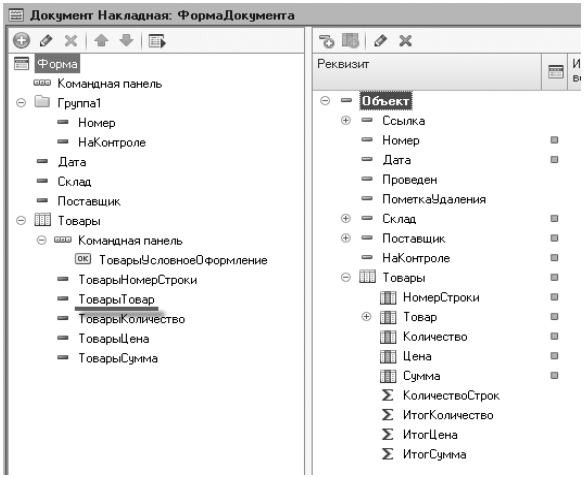


Рис. 3.116. В коллекции оформляемых полей указывается имя поля таблицы

Теперь запустим конфигурацию в режиме 1С:Предприятие и откроем накладную № 6. Ее табличная часть не содержит никакого условного оформления (рис. 3.117).

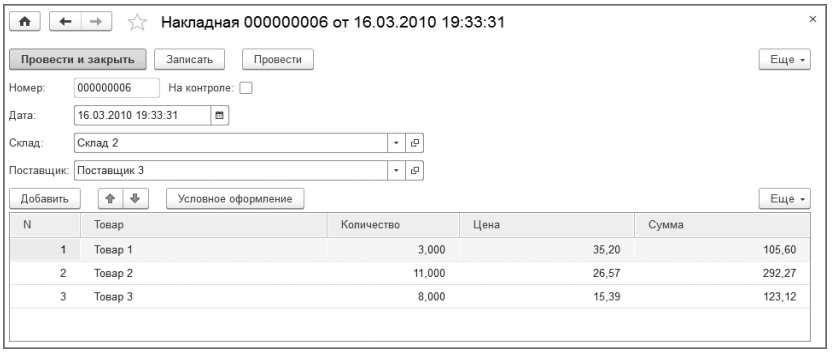


Рис. 3.117. Табличная часть без условного оформления

После того как мы выполним команду Условное оформление, товар Товар 2 будет выделен цветом фона, потому что количество этого товара больше 10 (рис. 3.118).

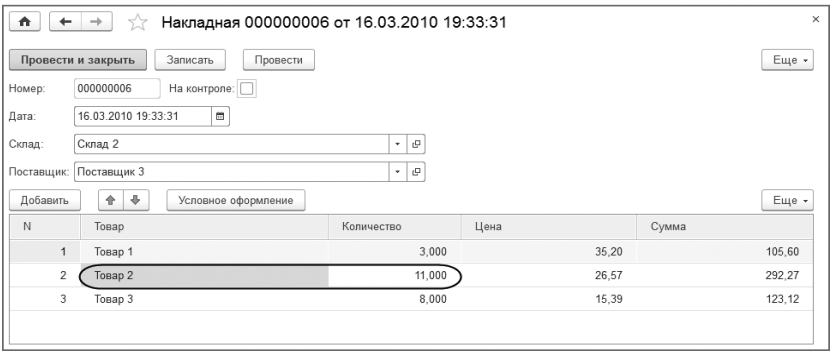


Рис. 3.118. Оформленная табличная часть

Пользователь не имеет стандартной возможности изменять условное оформление формы, поэтому в данной ситуации он не сможет отключить или удалить условное оформление, добавленное нами.

Напоследок хочется сделать еще одно небольшое замечание. В отличие от динамических списков, в условном оформлении формы обязательно нужно указывать оформляемые поля. Если хочется выделить всю строку табличной части, нужно указать все поля, которые в ней содержатся.

Глава 3.16. Дополнительные колонки в списках

Одна из часто встречающихся задач – добавить собственные колонки с дополнительными данными в динамический список или в табличную часть документа.

Обе эти задачи решаются относительно просто. В случае с динамическим списком нужно использовать произвольный запрос, который выведет требуемые поля данных. В случае с табличной частью решение может быть более сложным, и оно зависит от того, какие именно данные хочется показать в дополнительных колонках.

Динамический список

Для примера возьмем стандартную форму списка товаров, созданную конструктором. В этой форме находится динамический список, отображающий товары и некоторые их реквизиты (рис. 3.119).

Пример можно посмотреть в демонстрационной базе «Дополнительные колонки в списках», форма списка справочника Товары.

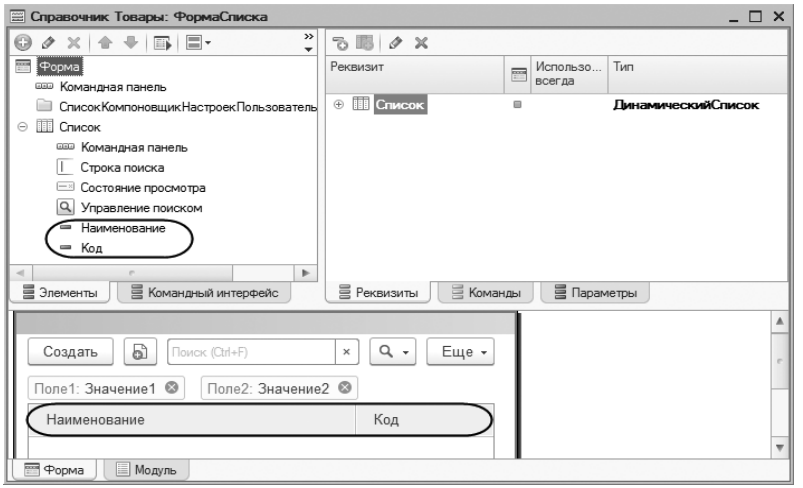


Рис. 3.119. Динамический список в форме

В списке отображаются только два реквизита справочника: Наименование и Код, хотя на самом деле справочник содержит также реквизиты Артикул и Сорт (рис. 3.120).

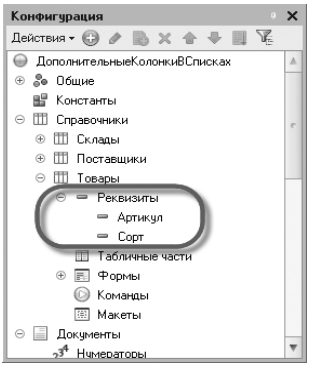


Рис. 3.120. Реквизиты справочника «Товары»

Это стандартное поведение конструктора. Он включает в список только необходимые реквизиты справочника. Другие реквизиты разработчик может добавить самостоятельно, если в этом есть необходимость.

У нас как раз есть такая необходимость. Мы хотим дополнительно отображать в этом списке колонки Артикул и Сорт. Поскольку эти поля являются реквизитами справочника, отобразить их в списке будет очень просто.

Если мы раскроем реквизит формы Список, то увидим, что нужные нам поля уже есть в составе этого реквизита (рис. 3.121).

Платформа, создавая динамический список, включила в него все поля, связанные со справочником Товары, но только два поля поместила в таблицу формы: Наименование и Код.

Поэтому наша задача решается очень просто. Нужно перетащить мышью поля Артикул и Сорт в дерево элементов, в таблицу Список (рис. 3.122).

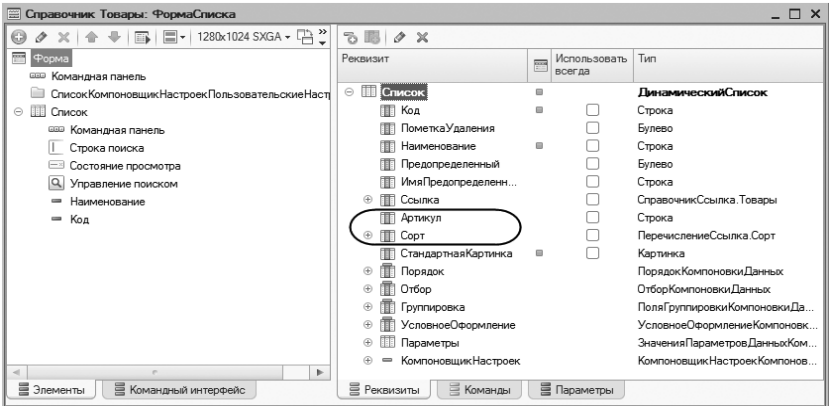


Рис. 3.121. Поля динамического списка

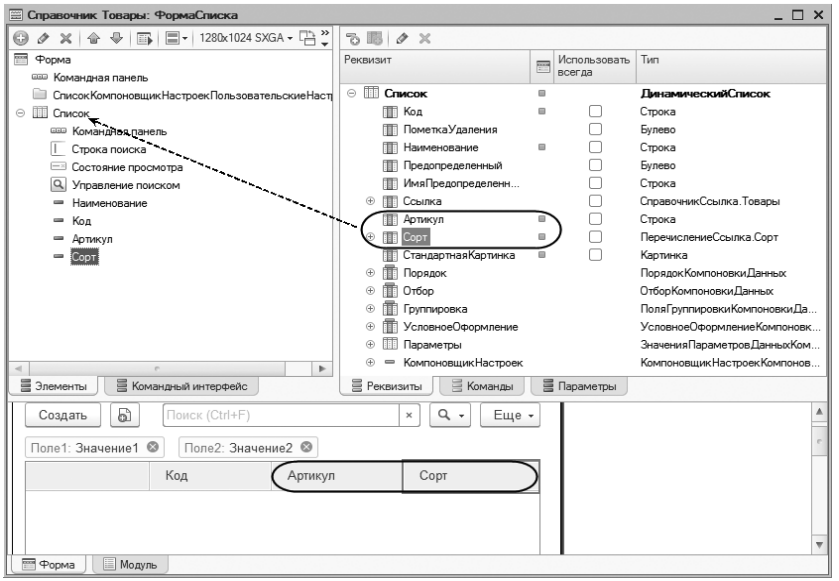


Рис. 3.122. Добавление колонок в список

Если теперь запустить конфигурацию в режиме 1С:Предприятие, мы увидим, что в списке товаров отображаются две дополнительные колонки Артикул и Сорт (рис. 3.123).

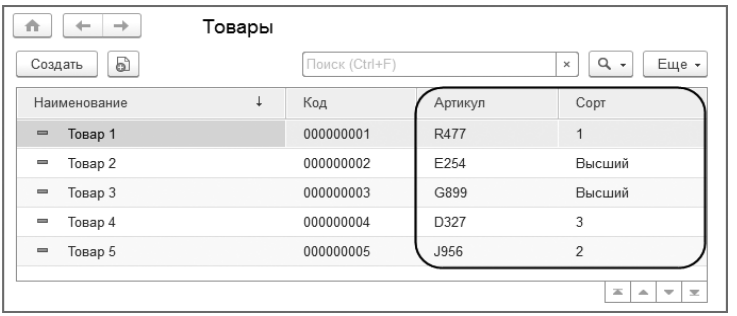


Рис. 3.123. Новые колонки в списке

Итак, мы рассмотрели простой случай, когда в динамический список нужно добавить дополнительные колонки, отображающие данные реквизитов.

Теперь рассмотрим более сложный случай, когда дополнительные данные не являются реквизитами объекта, содержащегося в основной таблице динамического списка.

В демонстрационной базе существует периодический регистр сведений Цены. В нем хранятся цены на товары (рис. 3.124).

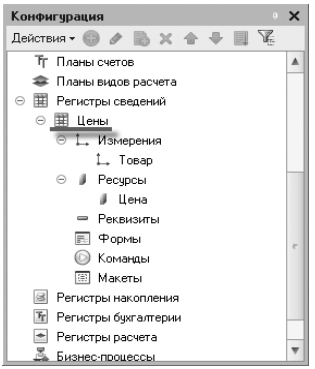


Рис. 3.124. Структура регистра «Цены»

Теперь наша задача будет заключаться в том, чтобы в списке товаров показать в отдельной колонке актуальную цену товара.

Для этого нам понадобится изменить текст запроса, который содержится в динамическом списке. В свойствах реквизита формы, содержащего динамический список, установим флажок Произвольный запрос (рис. 3.125).

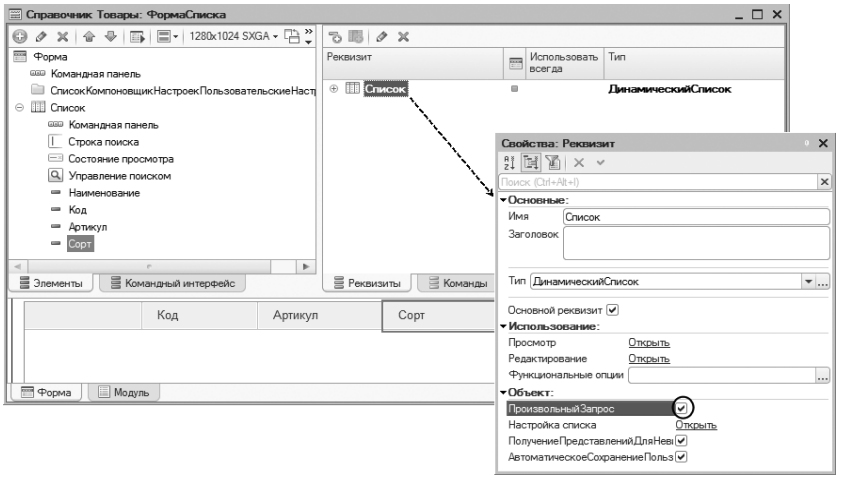


Рис. 3.125. Свойство «Произвольный запрос»

После этого состав свойств этого реквизита изменится. Нажав на гиперссылку Настройка списка, мы увидим стандартный текст запроса и сможем его изменить (рис. 3.126).

Заменим существующий текст запроса следующим (листинг 3.106).

Листинг 3.106. Произвольный запрос динамического списка

```
ВЫБРАТЬ
    СправочникТовары.Ссылка,
    СправочникТовары.ПометкаУдаления,
    СправочникТовары.Код,
    СправочникТовары.Наименование,
    СправочникТовары.Артикул,
    СправочникТовары.Сорт,
    СправочникТовары.Предопределенный,
    СправочникТовары.ИмяПредопределенныхДанных,
    ЦеныСрезПоследних.Цена
ИЗ
    РегистрСведений.Цены.СрезПоследних КАК ЦеныСрезПоследних
        ЛЕВОЕ СОЕДИНЕНИЕ Справочник.Товары КАК СправочникТовары
        ПО ЦеныСрезПоследних.Товар = СправочникТовары.Ссылка
```

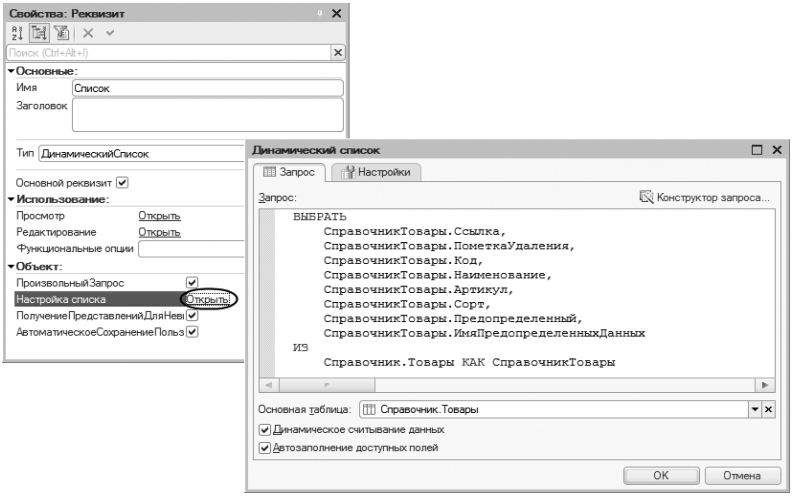


Рис. 3.126. Настройка динамического списка

Этим запросом мы дополнительно выводим поле Цена, соединяясь с виртуальной таблицей среза последних этого регистра.

Нажмем OK в диалоге Динамический список, вернемся к форме и развернем реквизит формы Список. Теперь среди его полей появилось поле Цена. Как и раньше, просто перетащим его в таблицу формы (рис. 3.127).

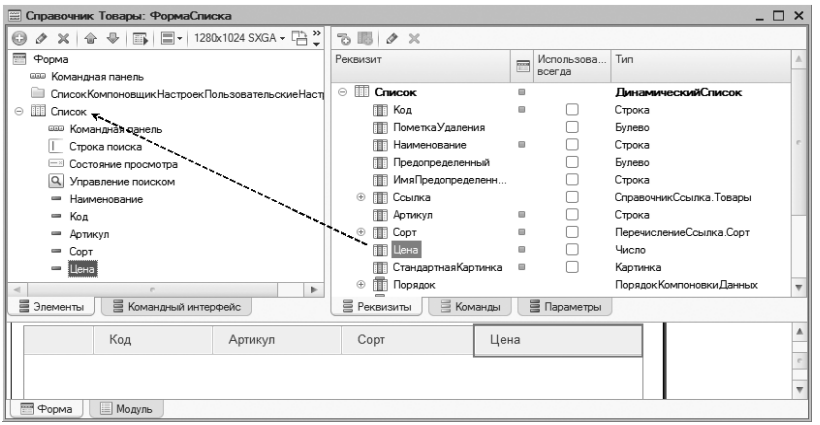


Рис. 3.127. Добавление нового поля

Теперь осталось запустить конфигурацию в режиме 1С:Предприятие и убедиться в том, что в списке товаров отображается еще одна дополнительная колонка, содержащая актуальную цену товара (рис. 3.128).

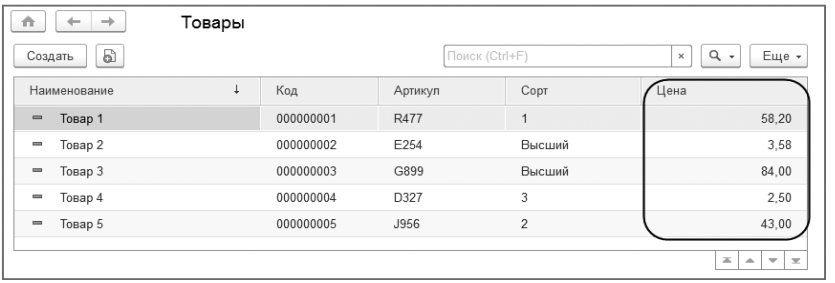


Рис. 3.128. Новая колонка в динамическом списке

Дополнительная обработка данных, получаемых динамическим списком

Иногда в динамическом списке требуется отобразить некоторую вспомогательную информацию, которую затруднительно, неэффективно или невозможно получить в рамках запроса динамического списка.

> Пример можно посмотреть в демонстрационной базе «Дополнительные колонки в списках», форма списка справочника Товары.

В этом случае можно воспользоваться обработчиком ПриПолученииДанныхНаСервере таблицы формы, отображающей данные динамического списка. Это внеконтекстный серверный обработчик, который позволяет выполнить дополнительную обработку данных, которые получает динамический список для отображения.

Данное событие вызывается после получения данных динамическим списком в том случае, если динамическим списком получена хотя бы одна строка. В обработчик события ПриПолученииДанныхНаСервере() первым параметром передается имя таблицы формы, из которой вызван этот обработчик.

Во втором параметре передается копия полных настроек динамического списка типа НастройкиКомпоновкиДанных. С помощью

свойства настроек ДополнительныеСвойства предоставляется возможность передавать необходимые данные из формы во внеконтекстный обработчик.

Третий параметр содержит коллекцию строк, которые будет отображать динамический список. В коллекции содержатся собственно данные (кроме строк группировок) и оформление этих данных.

Необходимо учитывать, что настройки списка: отбор (в том числе поиск), сортировка, группировка, условное оформление – будут применяться к значениям, полученным до изменения их обработчиком.

Для пояснения сказанного рассмотрим пример. Допустим, в списке товаров в отдельной колонке нам нужно выводить суммарное количество поступлений товаров за период, указанный в реквизите формы. В реальной жизни для этого, конечно, может использоваться регистр накопления остатков, но в нашей конфигурации его нет. Поэтому, в качестве примера, «оторванного от жизни», нам эта задача вполне подходит.

Откроем форму списка справочника Товары, добавим реквизит формы Период типа СтандартныйПериод и перетащим его в форму. Создадим у получившегося поля обработчик события ПриИзменении и заполним его следующим образом (листинг 3.107).

Листинг 3.107. Обработчик события «ПриИзменении» поля «Период»

```
&НаКлиенте
Процедура ПериодПриИзменении(Элемент)

    Список.КомпоновщикНастроек.Настройки.ДополнительныеСвойства.Вставить("ДатаНачала",
        Период.ДатаНачала);
    Список.КомпоновщикНастроек.Настройки.ДополнительныеСвойства.Вставить("ДатаОкончания",
        Период.ДатаОкончания);
    Элементы.Список.Обновить();

КонецПроцедуры
```

При изменении периода, за который нужно получить данные, мы сохраняем даты начала и окончания стандартного периода в структуре дополнительных свойств настроек динамического списка. И затем обновляем таблицу формы, отображающую данные списка.

При этом сначала выполняется запрос, указанный в свойствах динамического списка. Добавим туда заготовку для поля Поступило, в котором будет отображаться количество поступлений товара за заданный период (листинг 3.108).

Листинг 3.108. Произвольный запрос динамического списка

```
ВЫБРАТЬ
	СправочникТовары.Ссылка,
	СправочникТовары.ПометкаУдаления,
	СправочникТовары.Код,
	СправочникТовары.Наименование,
	СправочникТовары.Артикул,
	СправочникТовары.Сорт,
	СправочникТовары.Предопределенный,
	СправочникТовары.ИмяПредопределенныхДанных,
	ЦеныСрезПоследних.Цена,
	ВЫРАЗИТЬ (NULL КАК Число) КАК Поступило
ИЗ
	РегистрСведений.Цены.СрезПоследних КАК ЦеныСрезПоследних
		ЛЕВОЕ СОЕДИНЕНИЕ Справочник.Товары КАК СправочникТовары
		ПО ЦеныСрезПоследних.Товар = СправочникТовары.Ссылка
```

После получения данных запросом динамического списка вызывается событие ПриПолученииДанныхНаСервере таблицы формы, отображающей данные списка. В обработчике этого события мы и будем получать дополнительные данные и заполнять ими колонку списка Поступило.

Создадим обработчик события ПриПолученииДанныхНаСервере (обработчик должен быть внеконтекстным серверным) у таблицы формы Список и заполним его следующим образом (листинг 3.109).

Листинг 3.109. Обработчик события «ПриПолученииДанныхНаСервере» таблицы «Список»

```
&НаСервереБезКонтекста
Процедура СписокПриПолученииДанныхНаСервере(ИмяЭлемента, Настройки, Строки)

	Если Настройки.ДополнительныеСвойства.Свойство("ДатаНачала") И
			Настройки.ДополнительныеСвойства.Свойство("ДатаОкончания") Тогда
		ДатаНачала = Настройки.ДополнительныеСвойства.ДатаНачала;
		ДатаОкончания = Настройки.ДополнительныеСвойства.ДатаОкончания;
	КонецЕсли;
	Если НЕ ЗначениеЗаполнено(ДатаНачала) ИЛИ НЕ ЗначениеЗаполнено(ДатаОкончания) Тогда
		Возврат;
	КонецЕсли;

	Запрос = Новый Запрос;
```

```
    Запрос.Текст =
        "ВЫБРАТЬ
        |    МАКСИМУМ(НакладнаяТовары.Ссылка) КАК Ссылка,
        |    СУММА(НакладнаяТовары.Количество) КАК ИтогКоличество,
        |    НакладнаяТовары.Товар КАК Товар
        |ИЗ
        |    Документ.Накладная.Товары КАК НакладнаяТовары
        |ГДЕ
        |    НакладнаяТовары.Ссылка.Дата МЕЖДУ &ДатаНачала И &ДатаОкончания
        |    И    НакладнаяТовары.Товар В(&Товары)
        |    И    НакладнаяТовары.Ссылка.Проведен = ИСТИНА
        |
        |СГРУППИРОВАТЬ ПО
        |    НакладнаяТовары.Товар";

    Запрос.УстановитьПараметр("Товары", Строки.ПолучитьКлючи());
    Запрос.УстановитьПараметр("ДатаНачала", ДатаНачала);
    Запрос.УстановитьПараметр("ДатаОкончания", ДатаОкончания);

    Выборка = Запрос.Выполнить().Выбрать();
    Пока Выборка.Следующий() Цикл
        СтрокаСписка = Строки[Выборка.Товар];
        СтрокаСписка.Данные["Поступило"] = Выборка.ИтогКоличество;
        СтрокаСписка.Оформление["Поступило"].УстановитьЗначениеПараметра("ЦветТекста",
            WebЦвета.Малиновый);
    КонецЦикла;

КонецПроцедуры
```

Сначала мы получаем из структуры дополнительных свойств настроек динамического списка, переданных в параметре `Настройки`, даты начала и окончания периода, заданные в реквизите формы `Период`. Эти даты будут использоваться в качестве значений параметров `ДатаНачала` и `ДатаОкончания` запроса для получения данных.

С помощью этого запроса выбираются данные из табличных частей накладных за период, группируются по товарам, а значение поля `Количество` суммируется по каждому товару.

Массив выбираемых товаров ограничивается с помощью параметра `Товары`. Значение этого параметра получается с помощью метода `ПолучитьКлючи()` коллекции строк динамического списка, переданной в параметре `Строки`. Этот метод возвращает массив значений ключей для всех записей, которые будут отображаться динамическим списком.

В цикле обхода результата выполнения запроса из коллекции `Строки` по ключу товара мы получаем соответствующую ему строку динами-

ческого списка. И устанавливаем у этой строки в колонке Поступило (с помощью свойств `Данные` и `Оформление`) итоговое количество поступлений товара, а также выделяем это количество особым цветом текста.

Надо заметить, что работа с оформлением ячеек динамического списка (`Строки.Оформление`) ничем не отличается от работы с оформлением ячеек системы компоновки данных. Мы изменили цвет текста колонки Поступило просто в целях демонстрации.

После завершения обработчика измененный набор данных (коллекция `Строки`) будет передан клиентскому приложению для отображения таблицей формы, которая связана с динамическим списком.

Проверим, как это работает.

Раскроем реквизит Список и перетащим новое поле Поступило в таблицу формы (рис. 3.129).

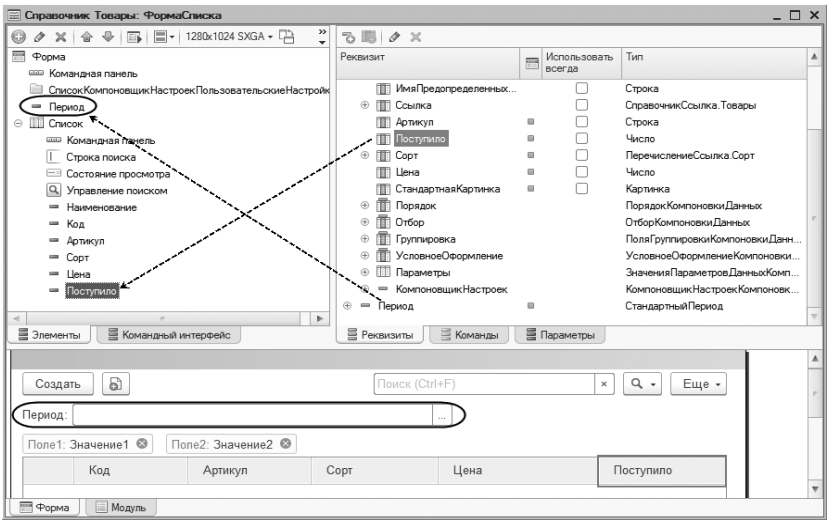


Рис. 3.129. Добавление новой колонки в список в конфигураторе

Запустим конфигурацию в режиме 1С:Предприятие. Сначала колонка Поступило в списке товаров пуста, так как поле Период еще не задано. Укажем период, и колонка Поступило сразу же заполнится итоговым

количеством поступивших товаров по накладным за указанный период. Причем данные в этой колонке будут выделены малиновым цветом текста (рис. 3.130).

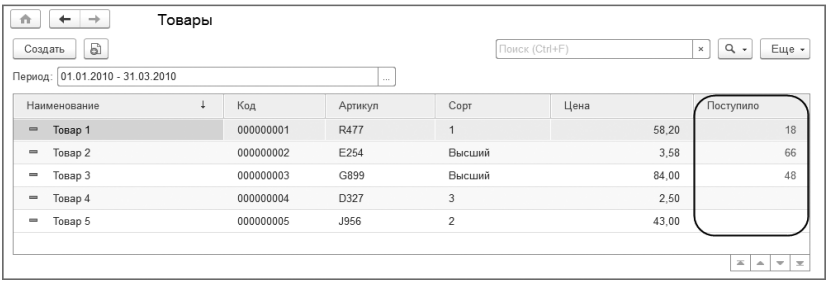


Рис. 3.130. Заполнение новой колонки в динамическом списке в режиме «1С:Предприятие»

Если настроить список программно или интерактивно (например, задать отбор, условное оформление и т. д.), то настройки списка будут применены, но только к данным, полученным до изменения их обработчиком `ПриПолученииДанныхНаСервере()`.

Необходимо заметить, что данные, передаваемые в этот обработчик события, уже обработаны условным оформлением динамического списка. Поэтому если требуется изменять оформление данных списка в зависимости от результата работы обработчика `ПриПолученииДанныхНаСервере`, то следует менять условное оформление непосредственно в самом обработчике.

Табличная часть

Добавление дополнительных колонок в табличную часть также рассмотрим на двух примерах. «Экспериментировать» будем со стандартной формой документа Накладная. Его табличная часть содержит колонки Номер строки, Товар, Количество, Цена и Сумма (рис. 3.131).

> Пример можно посмотреть в демонстрационной базе «Дополнительные колонки в списках», форма документа Накладная.

Как мы помним из предыдущего примера, у товара есть реквизит Артикул, который нам тоже хотелось бы видеть в табличной части этого документа.

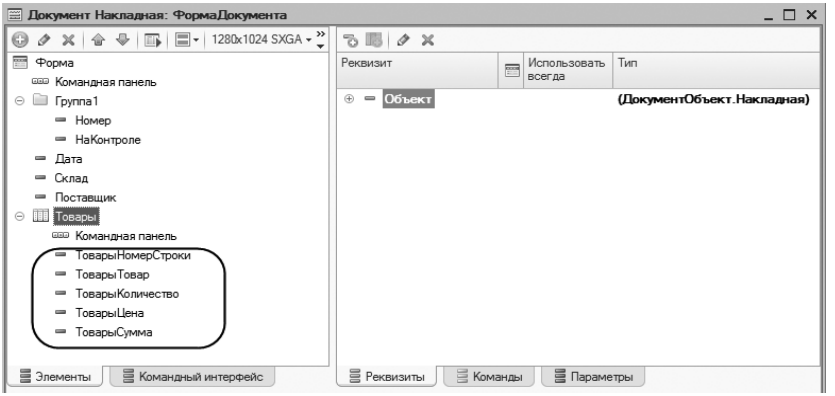


Рис. 3.131. Поля табличной части

Чтобы добавить Артикул, развернем реквизит формы Объект, развернем его табличную часть Товары, развернем реквизит Товар и увидим интересующее нас поле Артикул. Перетащим его в таблицу формы Товары (рис. 3.132).

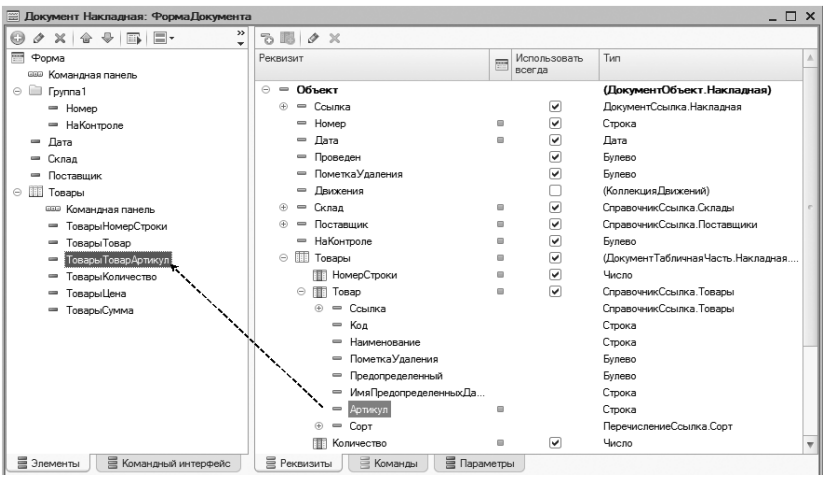


Рис. 3.132. Добавление нового поля в таблицу

Теперь останется только запустить конфигурацию в режиме 1С:Предприятие и убедиться, что в табличной части накладной отображается колонка с артикулом товара (рис. 3.133).

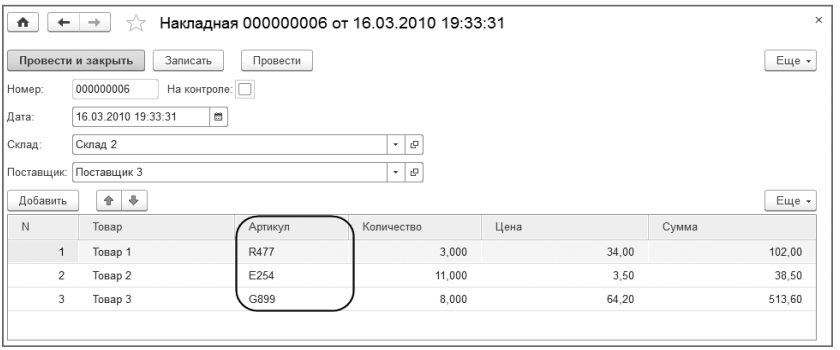


Рис. 3.133. Новая колонка в табличной части

Все просто, потому что добавляемые данные являются реквизитом товара, который сам по себе уже отображается в табличной части документа.

Но как быть в том случае, когда добавляемые данные не являются реквизитом, а должны быть получены в результате некоторых вычислений?

В этом случае все будет несколько сложнее. В реквизит формы, описывающий табличную часть документа, нужно будет добавить собственное поле и заполнить его данными в определенных обработчиках формы документа.

Чтобы не усложнять пример алгоритмами, не относящимися к сути, мы добавим в документ еще один реквизит товара – Сорт. Это позволит нам в обработчиках событий при заполнении собственной колонки данными просто писать `Товар.Сорт`. В реальной же ситуации тут будет некоторый алгоритм, который по имеющейся ссылке на товар будет получать данные, необходимые для отображения.

Итак, сначала в реквизит формы, в табличную часть Товары, добавим собственное поле СортТовара, имеющее тип ссылки на значение

перечисления Сорт. И затем перетащим это поле в таблицу формы Товары (рис. 3.134).

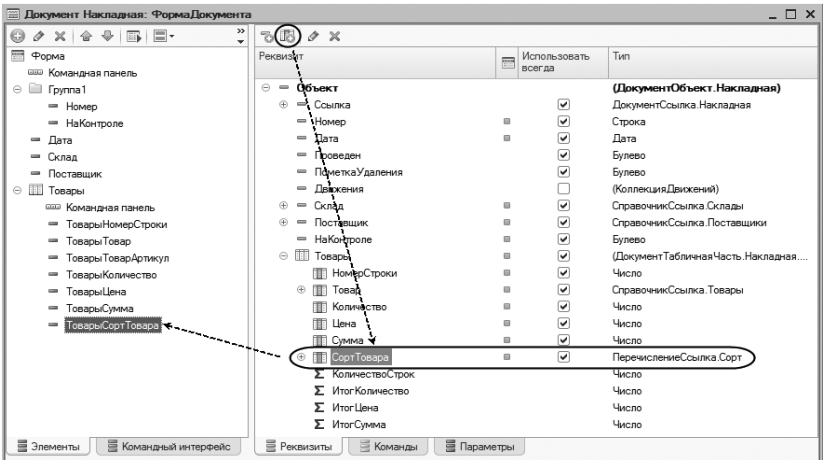


Рис. 3.134. Добавление реквизита формы и поля таблицы

Теперь начнем создавать обработчики событий, в которых колонка СортТовара будет заполняться данными.

Для того чтобы дополнительная колонка была заполнена данными при чтении существующей накладной из базы данных, создадим обработчик события формы При чтении на сервере (листинг 3.110).

Листинг 3.110. Обработчик события «При чтении на сервере»

```
// Заполнение дополнительной колонки при чтении существующего объекта
&НаСервере
Процедура ПриЧтенииНаСервере(ТекущийОбъект)

	// Алгоритм, по которому дополнительная колонка заполняется данными
	Для Каждого СтрокаДанных Из Объект.Товары Цикл
		СтрокаДанных.СортТовара = СтрокаДанных.Товар.Сорт;

	КонецЦикла

КонецПроцедуры
```

Для того чтобы поле дополнительной колонки было заполнено данными при изменении строки табличной части или при добавлении

новой строки, создадим обработчик события ПриИзменении поля таблицы ТоварыТовар. Поскольку это клиентский обработчик формы, для получения данных из базы данных будем вызывать серверную процедуру (листинг 3.111).

Листинг 3.111. Обработчик события «При изменении»

```
// Заполнение поля дополнительной колонки при изменении товара в табличной части
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

    // Алгоритм, по которому поле дополнительной колонки заполняется данными
    ДанныеСтроки = Элементы.Товары.ТекущиеДанные;
    ДанныеСтроки.СортТовара = ПолучитьСортТовара(ДанныеСтроки.Товар);

КонецПроцедуры

&НаСервереБезКонтекста
Функция ПолучитьСортТовара(Товар)

    Возврат Товар.Сорт;

КонецФункции
```

Для того чтобы дополнительная колонка была заполнена данными при копировании или при вводе накладной на основании другого объекта, создадим обработчик события формы При создании на сервере (листинг 3.112).

Листинг 3.112. Обработчик события «При создании на сервере»

```
// Заполнение дополнительной колонки при копировании или вводе на основании
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    // Алгоритм, по которому дополнительная колонка заполняется данными
    Для Каждого СтрокаДанных Из Объект.Товары Цикл

        Если НЕ ЗначениеЗаполнено(СтрокаДанных.СортТовара) Тогда
            СтрокаДанных.СортТовара = СтрокаДанных.Товар.Сорт;

        КонецЕсли;

    КонецЦикла

КонецПроцедуры
```

И, наконец, предусмотрим ситуацию, когда данные накладной могут быть изменены в процессе ее записи. На этот случай создадим обработчик события формы После записи на сервере (листинг 3.113).

Листинг 3.113. Обработчик события «После записи на сервере»

```
// Заполнение дополнительной колонки, если данные были изменены при записи.
&НаСервере
Процедура ПослеЗаписиНаСервере(ТекущийОбъект, ПараметрыЗаписи)

	// Алгоритм, по которому дополнительная колонка заполняется данными.
	Для Каждого СтрокаДанных Из Объект.Товары Цикл
		Если НЕ ЗначениеЗаполнено(СтрокаДанных.СортТовара) Тогда
			СтрокаДанных.СортТовара = СтрокаДанных.Товар.Сорт;

		КонецЕсли;

	КонецЦикла

КонецПроцедуры
```

Теперь осталось запустить конфигурацию в режиме 1С:Предприятие и убедиться, что при любых манипуляциях с накладной данные колонки Сорт заполняются правильно (рис. 3.135).

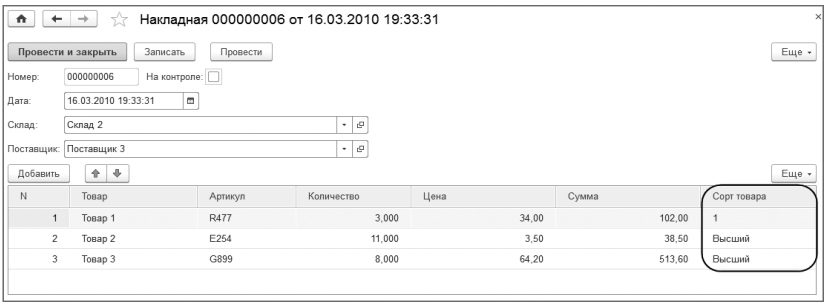


Рис. 3.135. Новая колонка в табличной части

Глава 3.17. Работа с таблицей в форме

При работе с табличной частью документа или просто произвольной таблицей значений, расположенной в форме, зачастую возникает необходимость выборочной или групповой обработки данных, содержащихся в таблице.

Особенность заключается в том, что для выполнения простейших действий с текущей строкой можно не обращаться к данным (реквизиту формы), а получить нужные значения прямо из элемента формы `Таблица`. Для этого используется свойство таблицы `ТекущиеДанные`.

Однако для выполнения более сложных или массовых операций требуется доступ непосредственно к данным формы, то есть к ее реквизиту. Реквизит формы может иметь разный тип – например, в случае табличной части это будет `ДанныеФормыКоллекция`, а в случае дерева значений – `ДанныеФормыДерево`. Сути дела это не меняет.

В любом случае реквизит формы является некоторой коллекцией. И когда мы «переходим» от элемента формы, в котором какая-то строка является текущей, к реквизиту, нужно уметь однозначно определить, какой элемент этой коллекции соответствует текущей строке таблицы. И наоборот, зная некоторый элемент коллекции, содержащейся в реквизите, нужно уметь сделать текущей строкой таблицы ту, которая соответствует этому элементу коллекции.

Для решения этой задачи используется *идентификатор* (рис. 3.136).

Находясь в таблице формы, узнать идентификатор текущей строки просто. Его возвращает свойство таблицы `ТекущаяСтрока`.

Имея этот идентификатор, найти соответствующий ему элемент коллекции, содержащейся в реквизите, тоже просто. У коллекции есть метод `НайтиПоИдентификатору()`, который вернет нужный элемент коллекции.

В «обратную сторону» все работает похожим образом (рис. 3.137).

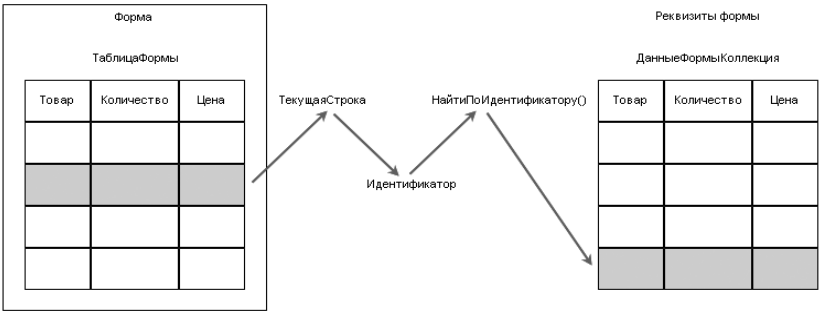


Рис. 3.136. Использование идентификатора для получения строки реквизита

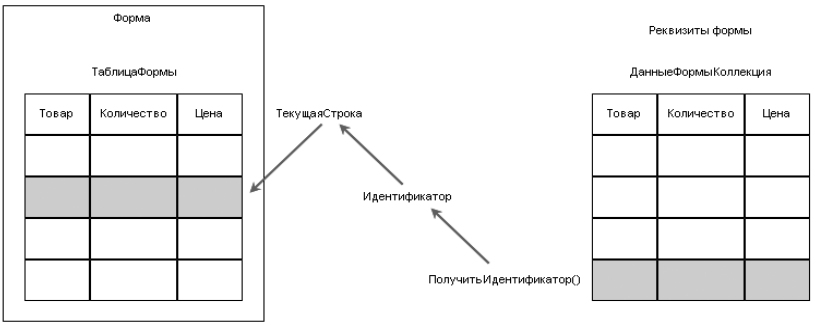


Рис. 3.137. Использование идентификатора для установки текущей строки таблицы

Имея некоторый элемент коллекции, содержащейся в реквизите, мы можем получить его идентификатор с помощью его метода `ПолучитьИдентификатор()`. Останется только присвоить этот идентификатор текущей строке таблицы, чтобы спозиционироваться на нужной строке.

Рассмотрим работу с таблицей на двух примерах. Первый будет заключаться в том, чтобы обойти поля таблицы в нужной последовательности, а второй – в том, чтобы сохранить текущую строку таблицы после массированного изменения данных.

Ввод данных по колонкам

Сразу оговоримся, что этот пример не совсем хорош. Мы будем использовать сообщения пользователю, но не по назначению.

Сообщения пользователю предназначены для того, чтобы прервать действие и указать на элемент формы, содержащий неправильные данные.

Мы же будем использовать их для того, чтобы «подсказывать» пользователю, в какой элемент формы необходимо ввести данные. Полезное свойство сообщений заключается в том, что если сообщение выводится с привязкой к элементу формы, то этот элемент сразу же переходит в режим редактирования. Таким образом, пользователю остается всего лишь ввести в него правильные данные.

Сценарий, который мы будем реализовывать, имеет под собой вполне реальную основу. Зачастую оператору, вводящему накладную, удобно вводить ее не по строкам, а по столбцам.

Сначала вводится вся номенклатура – первая колонка. Затем вводится все количество – вторая колонка. После этого вводится вся цена – третья колонка. В конце контролируется итоговая сумма, и если есть расхождения, то они исправляются путем построчной проверки сумм.

Такой сценарий удобен тем, что на каждом этапе выполняются однотипные действия, которые легко сверять с бумажным документом. Сначала – подбор номенклатуры. Затем – ввод чисел.

Пример можно посмотреть в демонстрационной базе «Таблица в форме», форма документа Накладная, обработчик ТоварыКоличествоПриИзменении() реквизита ТоварыКоличество таблицы.

Для того чтобы реализовать этот сценарий, напишем обработчик события При изменении для того реквизита таблицы, который предполагается заполнять по колонке.

В нашем случае таким реквизитом будет ТоварыКоличество (рис. 3.138).

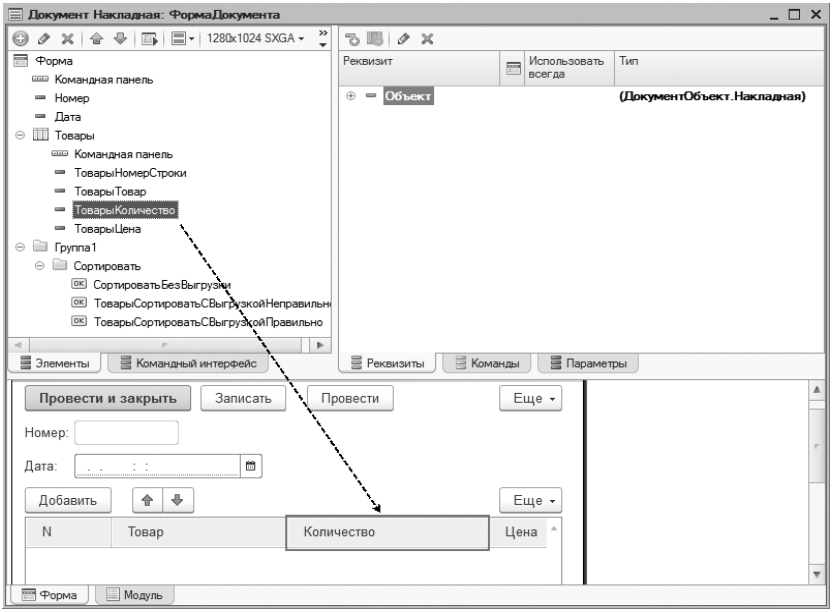


Рис. 3.138. Реквизит «ТоварыКоличество»

Обработчик будет выглядеть следующим образом (листинг 3.114).

Листинг 3.114. Обработчик события «При изменении»

```
СтрокаКоллекции = Объект.Товары.НайтиПоИдентификатору(Элементы.Товары.ТекущаяСтрока);
ИндексСтрокиКоллекции = Объект.Товары.Индекс(СтрокаКоллекции);

Если Объект.Товары.Количество() > ИндексСтрокиКоллекции + 1 Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.КлючДанных = Объект.Ссылка;
        Сообщение.ПутьКДанным = "Объект";
        Сообщение.Поле = "Товары[" + Строка(ИндексСтрокиКоллекции + 1) + "].Количество";
        Сообщение.Текст = "Введите количество товара";
        Сообщение.Сообщить();

КонецЕсли;
```

Сначала мы находим строку коллекции, которая соответствует текущей строке таблицы (листинг 3.115).

Листинг 3.115. Получение строки коллекции

```
СтрокаКоллекции = Объект.Товары.НайтиПоИдентификатору(Элементы.Товары.ТекущаяСтрока);
```

Затем получаем индекс этой строки в коллекции, т. к. нам нужно будет привязать сообщение к следующей строке таблицы. Следующая строка будет иметь индекс на единицу больше, чем текущая (листинг 3.116).

Листинг 3.116. Получение индекса строки коллекции

```
ИндексСтрокиКоллекции = Объект.Товары.Индекс(СтрокаКоллекции);
```

После этого мы формируем сообщение пользователю и привязываем его к полю Количество, находящемуся в следующей строке таблицы (листинг 3.117).

Листинг 3.117. Привязка сообщения к полю «Количество»

```
Сообщение.Поле = "Товары[" + Строка(ИндексСтрокиКоллекции + 1) + "].Количество";
```

Условие Если... нужно для того, чтобы не формировать сообщение в том случае, когда редактируется последняя строка таблицы (листинг 3.118).

Листинг 3.118. Контроль окончания редактируемой таблицы

```
Если Объект.Товары.Количество() > ИндексСтрокиКоллекции + 1 Тогда
```

В результате, если запустить систему в режиме 1С:Предприятие и заполнить номенклатуру в документе Накладная, мы сможем затем последовательно ввести количество номенклатуры, просто набирая его на клавиатуре и нажимая клавишу Ввод. Позиционирование на нужном элементе управления и вход в режим редактирования будут выполняться автоматически. Кроме этого, сообщение в форме всегда будет подсказывать текущую позицию ввода, что удобно для сверки с документом (рис. 3.139).

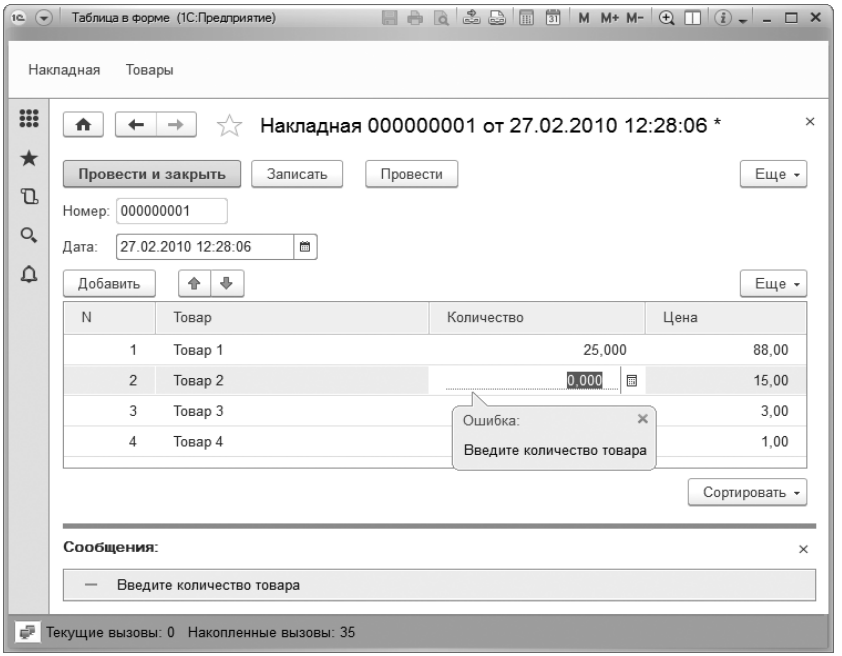


Рис. 3.139. Позиционирование на следующем поле и подсказка

Сохранение текущей строки после загрузки данных

Второй пример поможет понять некоторые особенности работы с коллекцией, которая хранит данные таблицы в реквизите формы. Мы будем рассматривать ситуацию, когда эти данные обрабатываются по некоторому непростому алгоритму. В примере мы будем всего лишь сортировать данные, но в реальной жизни это может быть гораздо более сложный алгоритм обработки.

Итак, наша задача – отсортировать табличную часть документа по возрастанию значений в колонке Цена.

Пример можно посмотреть в демонстрационной базе «Таблица в форме», форма документа Накладная, команды Сортировать без выгрузки, Сортировать с выгрузкой (неправильно), Сортировать с выгрузкой (правильно).
Это локальные команды формы.

Самый простой способ – обратиться непосредственно к реквизиту формы и выполнить его метод `Сортировать()` – листинг 3.119.

Листинг 3.119. Сортировка таблицы реквизита

```
&НаКлиенте
Процедура СортироватьБезВыгрузки(Команда)

    Объект.Товары.Сортировать("Цена");

КонецПроцедуры
```

Допустим, перед выполнением этого действия текущей была, например, третья строка табличной части (рис. 3.140).

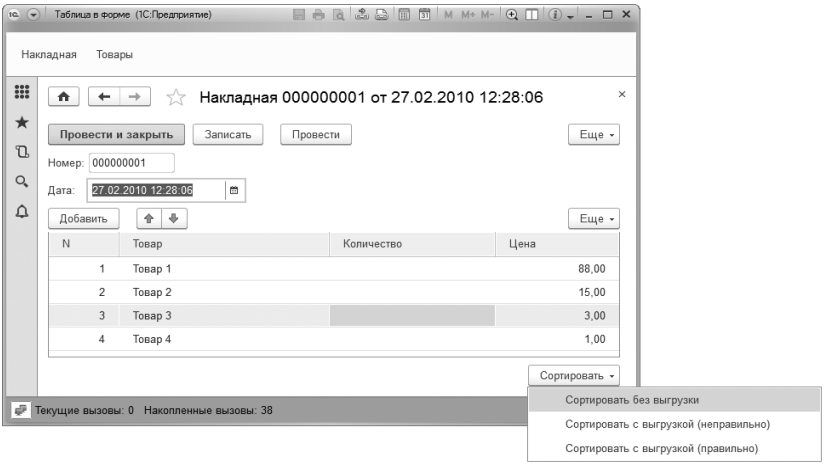


Рис. 3.140. Текущая строка перед выполнением сортировки

Тогда после выполнения сортировки эта же строка останется текущей. Она уже не будет третьей, но это будет та же самая строка (рис. 3.141).

Теперь рассмотрим другой случай. Допустим, для выполнения нашего алгоритма нужно выгрузить данные из реквизита в таблицу значений, обработать их и затем загрузить обратно (команда Сортировать с выгрузкой (неправильно)) – листинг 3.120.

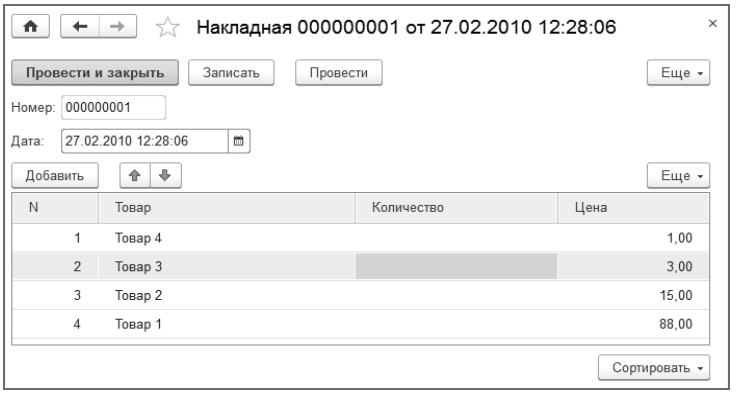


Рис. 3.141. Текущая строка после сортировки

Листинг 3.120. Сортировка с выгрузкой и загрузкой. Неправильный вариант

```
&НаКлиенте
Процедура СортироватьСВыгрузкойНеправильно(Команда)

	НаСервереНеправильно();

КонецПроцедуры

&НаСервере
Процедура НаСервереНеправильно()

	ТЗ = Объект.Товары.Выгрузить();
	ТЗ.Сортировать("Цена");
	Объект.Товары.Загрузить(ТЗ);

КонецПроцедуры
```

В этом случае результат окажется другим. Строки, как и раньше, будут отсортированы, однако текущей станет первая строка таблицы (рис. 3.142).

Так происходит потому, что после загрузки данных в реквизит формы в нем фактически оказывается новая коллекция данных. Они могут не совпадать с теми данными, которые были из реквизита выгружены. Поэтому текущей становится первая строка таблицы, что соответствует стандартному поведению платформы.

Однако мы знаем, что данные, которые загружены, те же самые, поменялся лишь порядок строк. Как в этом случае сохранить правильную текущую строку?

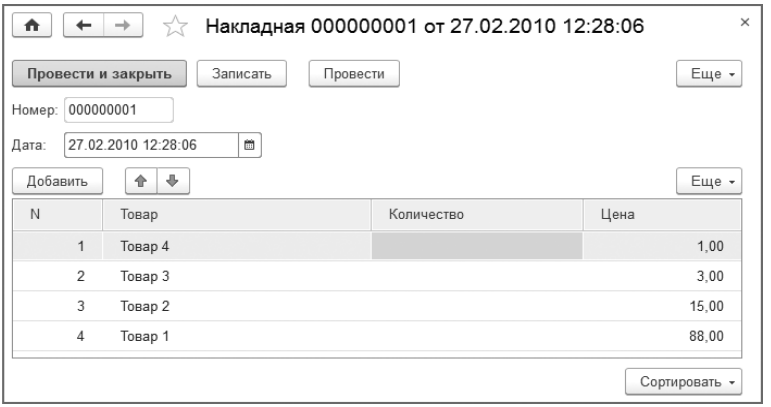


Рис. 3.142. Сброс текущей строки после сортировки

Рассмотрим третий вариант (команда Сортировать с выгрузкой (правильно)) – листинг 3.121.

Листинг 3.121. Сортировка с выгрузкой и загрузкой. Правильный вариант

```
&НаКлиенте
Процедура СортироватьСВыгрузкойПравильно(Команда)

	НаСервереПравильно();

КонецПроцедуры

&НаСервере
Процедура НаСервереПравильно()

	СтрокаКоллекции = Объект.Товары.НайтиПоИдентификатору(Элементы.Товары.ТекущаяСтрока);
	ИндексСтрокиКоллекции = Объект.Товары.Индекс(СтрокаКоллекции);

	ТЗ = Объект.Товары.Выгрузить();
	СтрокаВыгруженнойТаблицы = ТЗ.Получить(ИндексСтрокиКоллекции);

	ТЗ.Сортировать("Цена");

	ИндексСтрокиПослеСортировки = ТЗ.Индекс(СтрокаВыгруженнойТаблицы);

	Объект.Товары.Загрузить(ТЗ);

	СтрокаКоллекции = Объект.Товары.Получить(ИндексСтрокиПослеСортировки);
	Элементы.Товары.ТекущаяСтрока = СтрокаКоллекции.ПолучитьИдентификатор();

КонецПроцедуры
```

Прежде всего, оказавшись на сервере, мы находим ту строку реквизита, которая соответствует текущей строке в таблице формы (листинг 3.122, рис. 3.143).

Листинг 3.122. Получение строки в коллекции реквизита

```
СтрокаКоллекции = Объект.Товары.НайтиПоИдентификатору(Элементы.Товары.ТекущаяСтрока);
```

Рис. 3.143. Получение строки в коллекции реквизита

Затем мы получаем индекс этой строки (листинг 3.123, рис. 3.144).

Листинг 3.123. Получение индекса строки в реквизите

```
ИндексСтрокиКоллекции = Объект.Товары.Индекс(СтрокаКоллекции);
```

Рис. 3.144. Получение индекса строки в реквизите

Этот индекс понадобится нам в дальнейшем, чтобы найти по нему эту же строку, но уже в таблице значений, в которую мы выгрузим данные (листинг 3.124).

Листинг 3.124. Получение строки в таблице значений

```
ТЗ = Объект.Товары.Выгрузить();
СтрокаВыгруженнойТаблицы = ТЗ.Получить(ИндексСтрокиКоллекции);
```

Поскольку в процессе выгрузки мы не преобразовывали таблицу (не изменяли порядок и количество строк), то наша строка в реквизите и аналогичная строка в выгруженной таблице значений будут иметь одинаковые индексы (рис. 3.145).

Рис. 3.145. Получение строки в таблице значений

Теперь, после того как в переменной `СтрокаВыгруженнойТаблицы` мы запомнили «текущую строку» таблицы значений, можно эту таблицу значений сортировать (листинг 3.125, рис. 3.146).

Листинг 3.125. Сортировка таблицы значений

```
ТЗ.Сортировать("Цена");
```

Рис. 3.146. Сортировка таблицы значений

В отсортированной таблице значений порядок строк поменялся, и нам нужно определить, какой же индекс стал теперь у нашей «текущей строки» (листинг 3.126, рис. 3.147).

Листинг 3.126. Получение индекса строки в таблице значений

```
ИндексСтрокиПослеСортировки = ТЗ.Индекс(СтрокаВыгруженнойТаблицы);
```

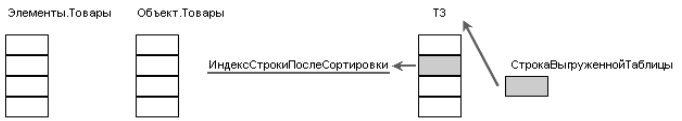


Рис. 3.147. Получение индекса строки в таблице значений

Зная этот индекс, мы можем загрузить данные обратно в реквизит и найти уже в нем ту строку, которая соответствует «текущей» (листинг 3.127, рис. 3.148).

Листинг 3.127. Получение строки в реквизите

```
Объект.Товары.Загрузить(ТЗ);
СтрокаКоллекции = Объект.Товары.Получить(ИндексСтрокиПослеСортировки);
```

Рис. 3.148. Получение строки в реквизите

И в заключение, зная «текущую строку» в реквизите, мы можем спозиционировать курсор в таблице формы на нужную нам строку, используя идентификатор (листинг 3.128, рис. 3.149).

Листинг 3.128. Позиционирование текущей строки в таблице формы

```
Элементы.Товары.ТекущаяСтрока = СтрокаКоллекции.ПолучитьИдентификатор();
```

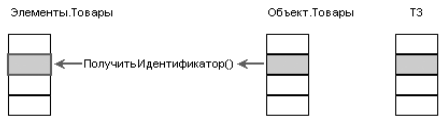


Рис. 3.149. Позиционирование текущей строки в таблице формы

Если теперь посмотреть, как это работает в режиме 1С:Предприятие, мы увидим, что после сортировки текущей остается та строка, которая была ею перед началом действия.

Глава 3.18. Работа с файлами и картинками

Тонкий клиент и веб-клиент имеют разные возможности по работе с файлами и картинками.

Поскольку веб-клиент является типичным веб-приложением, он ничего не знает о локальной файловой системе компьютера, на котором работает. Его собственные стандартные возможности заключаются лишь в том, чтобы поместить один файл на сервер или получить один файл из базы данных.

Причем и в том и в другом случае передается только один файл, и эта операция интерактивна в том смысле, что отсутствуют какие-либо средства программной работы с локальной файловой системой клиентского компьютера.

Исходный файл для помещения на сервер должен быть выбран пользователем интерактивно, или путь к нему должен быть указан в явном виде. Аналогично и при получении файла из базы данных – его конечное местоположение должно быть выбрано пользователем интерактивно или в явном виде указано в программном коде.

Тонкий клиент является Windows-приложением и имеет более широкие возможности работы с файлами. В частности, он поддерживает множественный экспорт и импорт файлов и предоставляет программные средства для работы с локальной файловой системой клиентского компьютера.

Для того чтобы веб-клиент смог выполнять такие же действия, на клиентском компьютере необходимо установить и подключить *расширение работы с файлами*. Это отдельный компонент платформы, его установка и подключение выполняются автоматически средствами встроенного языка. Но установка этого расширения должна быть интерактивной. Пользователь должен самостоятельно принять решение об установке.

Однако использование расширения работы с файлами снижает общую безопасность системы: в браузере Microsoft Internet Explorer требуется разрешение на установку и использование ActiveX, а в браузере Mozilla Firefox необходимо предоставление расширенных приви-

легий. Пользователь веб-клиента попросту может не иметь подобных прав, если работа выполняется на публичном компьютере или в организации, использующей определенную политику безопасности.

Поэтому при работе с файлами и картинками следует в обязательном порядке реализовывать алгоритмы, использующие стандартные, общие для тонкого и веб-клиента возможности работы с файлами. Кроме того, можно реализовать и другую ветку алгоритма, использующую расширенные возможности работы с файлами. Но эта ветка должна быть опциональной, дополнительной. Нельзя обязательно заставлять пользователя использовать расширение работы с файлами, ведь у него может не быть прав на его установку.

Стандартные возможности

Стандартные возможности работы с файлами реализуются двумя методами встроенного языка: `НачатьПомещениеФайла()` и `ПолучитьФайл()`.

`НачатьПомещениеФайла()` – помещает файл из локальной файловой системы во временное хранилище.

`ПолучитьФайл()` – получает файл из информационной базы и помещает его на клиентский компьютер.

Помещение файла в информационную базу выполняется в два этапа.

Сначала с помощью метода `НачатьПомещениеФайла()` файл с клиентского компьютера помещается во *временное хранилище* (рис. 3.150).

Временное хранилище – это набор служебных файлов кластера, в которых в течение ограниченного времени могут храниться произвольные данные до помещения их в базу данных.

На втором этапе, когда пользователь принимает решение сохранить редактируемые данные в базе данных, файл из временного хранилища помещается в какой-либо реквизит, имеющий тип `ХранилищеЗначения`, и эти данные записываются в базу данных (рис. 3.151).

Если же пользователь отказывается от сохранения редактируемых данных, то файл удаляется из временного хранилища либо автоматически, когда будет закрыта форма, с помощью которой он туда был помещен, либо «вручную», из встроенного языка.

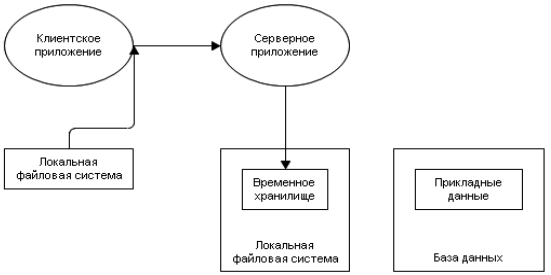


Рис. 3.150. Помещение файла во временное хранилище

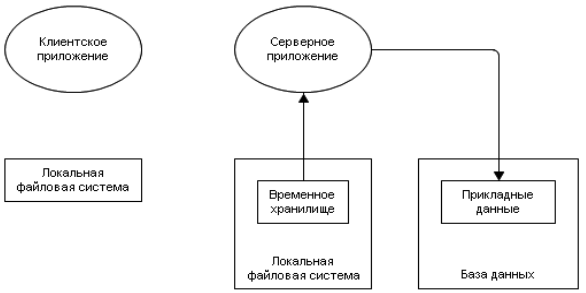


Рис. 3.151. Перемещение файла из временного хранилища в базу данных

Процесс получения файла из информационной базы проще. Он выполняется в один этап, с помощью метода `ПолучитьФайл()` (рис. 3.152).

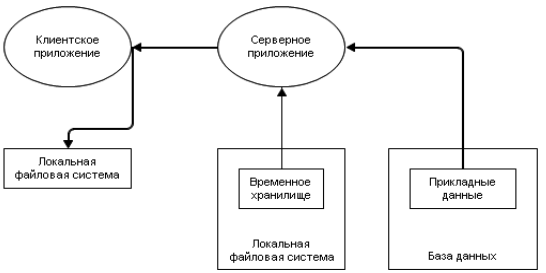


Рис. 3.152. Извлечение файла из временного хранилища или из базы данных

Файл извлекается из реквизита типа ХранилищеЗначения и помещается на компьютер пользователя.

С помощью этого метода можно получить на клиентский компьютер данные не только из базы данных, но и из временного хранилища. При работе с файлами эта возможность имеет малое практическое значение. Зато она востребована, например, в тех случаях, когда временное хранилище используется для передачи больших массивов данных между двумя формами на клиенте. Такой пример рассмотрен в разделе «Использование временного хранилища для передачи данных между формами» на стр. 831.

Расширенные возможности

Как уже говорилось раньше, в тонком клиенте расширенные возможности работы с файлами доступны всегда. В веб-клиенте для их использования нужно установить и подключить расширение работы с файлами.

Здесь мы лишь перечислим эти возможности, так как их практическое использование, как правило, не вызывает трудностей.

Установка и подключение расширения работы с файлами выполняются с помощью двух методов встроенного языка:

- НачатьУстановкуРасширенияРаботыСФайлами(),
- НачатьПодключениеРасширенияРаботыСФайлами().

После этого в веб-клиенте становятся доступны методы экспорта/импорта файлов:

- НачатьПомещениеФайлов(),
- НачатьПолучениеФайлов().

Они аналогичны упоминавшимся ранее методам, но позволяют выполнять операции сразу над группой файлов.

Также становятся доступны методы прямого доступа к локальной файловой системе клиентского компьютера:

- НачатьКопированиеФайла(),
- НачатьПоискФайлов(),
- НачатьПеремещениеФайла(),

- НачатьСозданиеКаталога(),
- НачатьУдалениеФайлов().

Чтобы не запутаться в различных методах работы с файлами, все их можно представить в виде следующей иерархии (рис. 3.153).

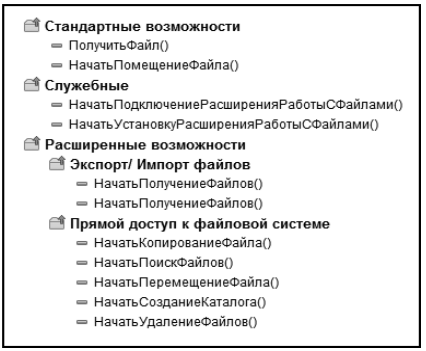


Рис. 3.153. Методы работы с файлами

Получение файла и сохранение его в базе данных

В качестве примера рассмотрим задачу получения файла с клиентского компьютера и сохранения его в реквизите справочника.

Пусть в конфигурации существует справочник Поставщики. Для каждого поставщика в базе данных должен храниться произвольный файл, содержащий договор с этим поставщиком.

Пример можно посмотреть в базе «Файлы и картинки», справочник Поставщики.

Сразу оговоримся, что рассматриваемый пример упрощен и произвольный файл мы будем хранить в одном из реквизитов самого справочника Поставщики. В реальной работе, как правило, так не поступают.

Для хранения больших объемов бинарных данных (файлы, картинки) создаются отдельные объекты конфигурации (справочники или регистры сведений), связанные с данным объектом. Такой подход

позволяет исключить обязательное считывание больших объемов информации при считывании самого объекта.

Итак, справочник Поставщики будет содержать два реквизита: ФайлДоговора (типа ХранилищеЗначения для хранения двоичных данных) и ИмяФайлаДоговора (типа Строка для хранения имени загруженного файла), рис. 3.154.

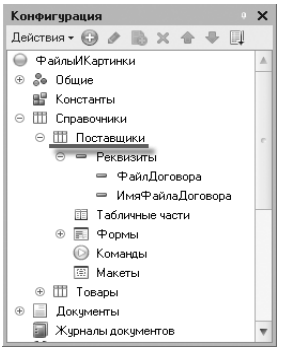


Рис. 3.154. Структура справочника «Поставщики»

Имя загруженного файла понадобится нам для того, чтобы иметь возможность в дальнейшем этот файл поместить обратно на компьютер пользователя.

Форма элемента справочника Поставщики помимо данных самого объекта будет содержать два дополнительных строковых реквизита: ИмяФайлаДоговора и ССылкаНаФайлВоВременномХранилище (рис. 3.155).

Ссылка на временное хранилище запоминается в форме потому, что она необходима лишь на момент от загрузки файла до записи объекта в базу данных и не является частью прикладных данных объекта.

ИмяФайлаДоговора запоминается в форме для того, чтобы не модифицировать данные объекта до тех пор, пока не начнется запись объекта в базу данных. В принципе, можно обойтись и без него – сохранять имя файла прямо в реквизит объекта, но тогда несколько сложнее станет проверка, выполняемая при сохранении файла на компьютер пользователя.

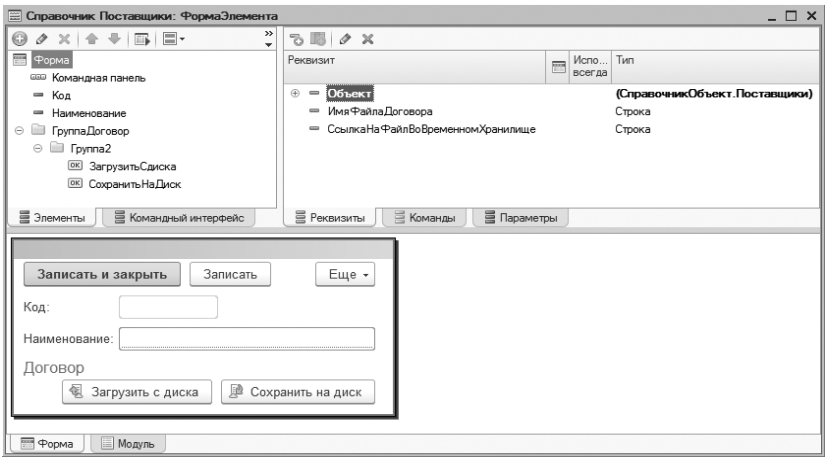


Рис. 3.155. Реквизиты формы элемента справочника «Поставщики»

Для загрузки файла с диска в информационную базу и для сохранения его на диск в форме созданы две локальные команды: ЗагрузитьСДиска и СохранитьНаДиск.

Загрузка файла выполняется с помощью немодального метода глобального контекста `НачатьПомещениеФайла()`. Первым параметром в этот метод передается описание оповещения, описывающее экспортную процедуру (расположенную в том же модуле – модуле формы) `ЗагрузитьСДискаЗавершение()`, которая будет вызвана, когда пользователь выберет файл (листинг 3.129).

В этой процедуре обработки оповещения в случае выбора файла пользователем и будут выполняться действия по загрузке файла с диска (листинг 3.130).

Листинг 3.129. Загрузка файла

```
&НаКлиенте
Процедура ЗагрузитьСДиска(Команда)

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьСДискаЗавершение",
        ЭтотОбъект), , , Истина , УникальныйИдентификатор);

КонецПроцедуры
```

Помимо описания оповещения в метод НачатьПомещениеФайла() четвертым параметром передается Истина – признак интерактивного выбора файла. И пятым параметром передается уникальный идентификатор открытой формы, о котором речь пойдет ниже.

Листинг 3.130. Обработчик оповещения «ЗагрузитьСДискаЗавершение»

```
&НаКлиенте
Процедура ЗагрузитьСДискаЗавершение(Результат, АдресВХранилище, ВыбранноеИмяФайла,
                                    ДополнительныеПараметры) Экспорт

    Если Результат Тогда

        Файл = Новый Файл(ВыбранноеИмяФайла);

        ИмяФайлаДоговора = Файл.Имя;
        ССылкаНаФайлВоВременномХранилище = АдресВХранилище;

        Модифицированность = Истина;
    КонецЕсли;

КонецПроцедуры
```

В процедуру, вызывающуюся после выбора файла, в параметре Результат передается признак того, что пользователь осуществил выбор файла. Потому что в процессе выбора файла пользователь может, вообще говоря, отказаться от задуманной операции – загрузки файла. Поэтому модификация реквизитов формы выполняется только в том случае, когда файл выбран успешно.

В параметре АдресВХранилище передается адрес во временном хранилище, по которому был помещен файл. И в параметре ВыбранноеИмяФайла передается путь к выбранному файлу.

В случае успешного выбора файла короткое имя файла помещается в реквизит формы ИмяФайлаДоговора, а ссылка на этот файл во временном хранилище помещается в реквизит формы ССылкаНаФайлВоВременномХранилище.

Для того чтобы из полного имени файла получить короткое имя, используется программный объект Файл (листинг 3.131).

Листинг 3.131. Получение короткого имени файла

```
Файл = Новый Файл(ВыбранноеИмяФайла);
ИмяФайлаДоговора = Файл.Имя;
```

Немного отступим от нашего примера, чтобы дать полезный совет. Быстро и без ошибок создать заготовку процедуры обработки оповещения и сконструировать объект `ОписаниеОповещения` при вызове немодального метода можно с помощью механизма рефакторинга. Для этого достаточно написать имя метода, поставить открывающую скобку (например, «`НачатьПомещениеФайла(`)»), установить курсор на имя метода и вызвать из контекстного меню команду Рефакторинг – Создать обработку оповещения. Затем в отдельном окне можно задать имя процедуры обработки оповещения или согласиться с тем именем, которое предлагает платформа (рис. 3.156).

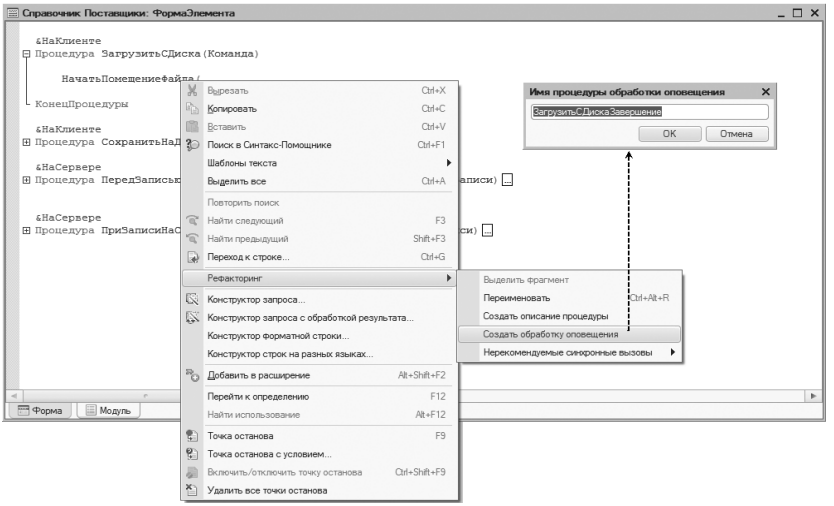


Рис. 3.156. Создание обработки оповещения с помощью рефакторинга

Вернемся к нашему примеру. Итак, после помещения файл окажется во временном хранилище, а в реквизитах формы будет ссылка на него и его имя (рис. 3.157).

Примечательным здесь является то, что при помещении файла мы указали последний параметр метода `НачатьПомещениеФайла()` (листинг 3.132).

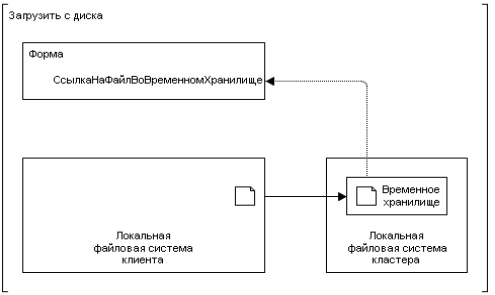


Рис. 3.157. Файл во временном хранилище

Листинг 3.132. Привязка файла к открытой форме

```
НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьСДискаЗавершение", ЭтотОбъект),
                     ,, Истина , УникальныйИдентификатор);
```

В качестве этого параметра мы передали уникальный идентификатор нашей открытой формы. Это значит, что файл во временном хранилище будет существовать до тех пор, пока мы не закроем форму. Тогда платформа автоматически удалит его из хранилища. Значит, если пользователь откажется от сохранения изменений, сделанных в форме, нам не нужно заботиться об удалении файла из временного хранилища – платформа это сделает сама.

А вот если пользователь решит сохранить сделанные изменения, тогда будут выполнены следующие действия.

Сначала на сервере будет вызван обработчик события формы Перед записью на сервере (листинг 3.133).

Листинг 3.133. Обработчик события «Перед записью на сервере»

```
&НаСервере
Процедура ПередЗаписьюНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    // Получить файл из хранилища и поместить его в объект.
    Если ЭтоАдресВременногоХранилища(ССылкаНаФайлВоВременномХранилище) Тогда
        ДвоичныеДанные =
            ПолучитьИзВременногоХранилища(ССылкаНаФайлВоВременномХранилище);
        ТекущийОбъект.ФайлДоговора =
            Новый ХранилищеЗначения(ДвоичныеДанные, Новый СжатиеДанных(9));
        ТекущийОбъект.ИмяФайлаДоговора = ИмяФайлаДоговора;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы поместим файл из временного хранилища в реквизит записываемого объекта, и в другой реквизит объекта поместим имя этого файла. Делать это мы будем не всегда (ведь пользователь может записать объект, и не указывая файл договора), а только в том случае, когда реквизит формы действительно хранит ссылку на файл во временном хранилище (листинг 3.134).

Листинг 3.134. Проверка того, что навигационная ссылка указывает на временное хранилище

```
Если ЭтоАдресВременногоХранилища(ССылкаНаФайлВоВременномХранилище) Тогда
```

В этом случае мы получим из временного хранилища двоичные данные и преобразуем их в объект `ХранилищеЗначения`, который и поместим в реквизит записываемого объекта (листинг 3.135).

Листинг 3.135. Помещение файла в реквизит объекта

```
ДвоичныеДанные = ПолучитьИзВременногоХранилища(ССылкаНаФайлВоВременномХранилище);
ТекущийОбъект.ФайлДоговора =
                Новый ХранилищеЗначения(ДвоичныеДанные, Новый СжатиеДанных(9));
```

Таким образом, после выполнения этого обработчика мы будем иметь следующую картину (рис. 3.158).

Рис. 3.158. Файл в реквизите объекта

Затем платформа начнет запись объекта, и его данные будут помещены в базу данных (рис. 3.159).

После этого, еще до окончания транзакции записи, на сервере будет вызван обработчик события формы `При записи на сервере` (листинг 3.136).

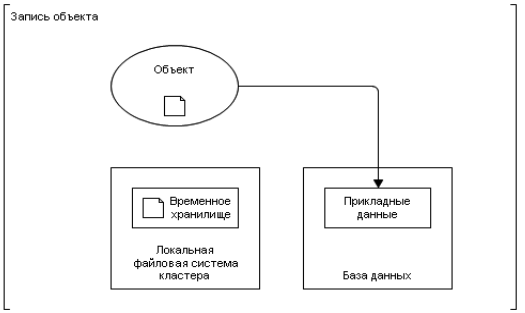


Рис. 3.159. Запись данных объекта в базу данных

Листинг 3.136. Обработчик события «При записи на сервере»

```
&НаСервере
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    // Удалить файл из временного хранилища
    Если ЭтоАдресВременногоХранилища(ССылкаНаФайлВоВременномХранилище) Тогда
        УдалитьИзВременногоХранилища(ССылкаНаФайлВоВременномХранилище);

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы «наведем красоту». Так как данные объекта уже записаны в базу данных, полученный файл успешно сохранен, можно удалить из временного хранилища находящийся там файл. Для этого мы используем ссылку, сохраненную в реквизите формы (рис. 3.160).

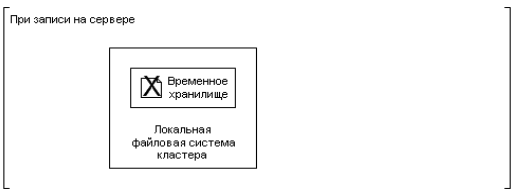


Рис. 3.160. Удаление файла из временного хранилища

Теоретически можно этого и не делать: платформа все равно автоматически удалит этот файл, когда мы закроем форму. Но в общем

случае при интенсивной многопользовательской работе лучше освобождать ресурсы сервера сразу же после того, как необходимость в них исчезает.

Процесс получения файла из информационной базы выглядит гораздо проще. Он целиком реализован в обработчике локальной команды формы СохранитьНаДиск (листинг 3.137).

Листинг 3.137. Получение файла из информационной базы

```
&НаКлиенте
Процедура СохранитьНаДиск(Команда)

    Если Объект.ИмяФайлаДоговора = "" Тогда
        ПоказатьПредупреждение(, "У поставщика нет сохраненного в базе договора");

    Иначе
        ССылкаНаФайлВИБ = ПолучитьНавигационнуюССылку(Объект.Ссылка, "ФайлДоговора");
        ПолучитьФайл(ССылкаНаФайлВИБ, Объект.ИмяФайлаДоговора);

    КонецЕсли;

КонецПроцедуры
```

Поскольку имя файла договора мы записываем в реквизит объекта только в момент записи, можно использовать его для определения того, есть ли в объекте сохраненный файл или нет.

Если нет, то показываем пользователю сообщение с помощью немодального метода `ПоказатьПредупреждение()`. Первый параметр – описание оповещения – в этом вызове опущен, так как после вывода предупреждения никаких действий выполнять не требуется.

Чтобы получить файл из информационной базы, нужно иметь навигационную ссылку на этот файл.

Навигационную ссылку мы получаем, указывая ссылку на объект и указывая имя реквизита объекта, в котором хранится файл (листинг 3.138).

Листинг 3.138. Получение навигационной ссылки на реквизит объекта

```
ССылкаНаФайлВИБ = ПолучитьНавигационнуюССылку(Объект.Ссылка, "ФайлДоговора");
```

Затем просто получаем файл по ссылке ССылкаНаФайлВИБ и сохраняем его на компьютер пользователя с именем, заданным в реквизите ИмяФайлаДоговора (листинг 3.139).

Листинг 3.139. Получение файла на компьютер пользователя

```
ПолучитьФайл(ССылкаНаФайлВИБ, Объект.ИмяФайлаДоговора);
```

При выполнении этого метода будет открыт стандартный диалог операционной системы, который предложит либо открыть, либо сохранить получаемый файл (рис. 3.161).

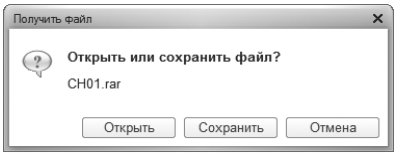


Рис. 3.161. Диалог получения файла

Картинка товара в форме

При работе с картинками используется тот же самый подход, который был описан выше. Отличие заключается только в том, что в предыдущем примере нам не нужно было никак отображать файл в форме элемента справочника Поставщики. А картинка, как правило, загружается в информационную базу именно для того, чтобы отображать ее в какой-либо форме.

Пример можно посмотреть в демонстрационной базе «Файлы и картинки», справочник Товары.

Работу с картинками рассмотрим на примере справочника Товары и картинки товара. Ее нужно загрузить, отобразить в форме, сохранить в базе данных и иметь возможность получить из базы данных на клиентский компьютер.

Снова сделаем оговорку. Рассматриваемый пример упрощен, и картинку мы будем хранить в одном из реквизитов самого справочника Товары. В реальной работе, как правило, так не поступают,

а хранят картинки в отдельных объектах конфигурации, в справочниках или в регистрах сведений.

Как и в предыдущем примере, справочник Товары будет содержать два реквизита: ФайлКартинки, чтобы хранить саму картинку, и ИмяФайлаКартинки, чтобы знать ее имя при выгрузке на клиентский компьютер (рис. 3.162).

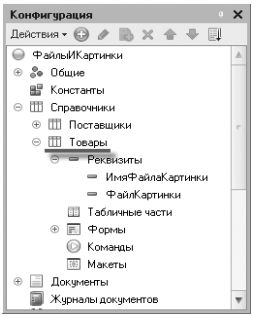


Рис. 3.162. Структура справочника «Товары»

Форма элемента справочника Товары также будет содержать два дополнительных реквизита: ИмяФайлаКартинки и ССылкаНаКартинку (рис. 3.163).

ИмяФайлаКартинки в форме нужно, чтобы не модифицировать данные объекта до того, как начнется запись в базу данных, а ССылкаНаКартинку будет содержать навигационную ссылку на картинку, находящуюся во временном хранилище или в базе данных. Этот реквизит связан с полем формы, имеющим вид Поле картинки. Таким образом, платформа будет автоматически отображать в форме ту картинку, которая находится по ссылке, содержащейся в этом реквизите.

Для загрузки картинки с диска в информационную базу и для сохранения ее на диск в форме созданы две локальные команды: ЗагрузитьСДиска и СохранитьНаДиск.

Загрузка картинки выполняется также с помощью немодального метода `НачатьПомещениеФайла()`, который мы подробно рассматривали выше на стр. 642 (листинги 3.140, 3.141).

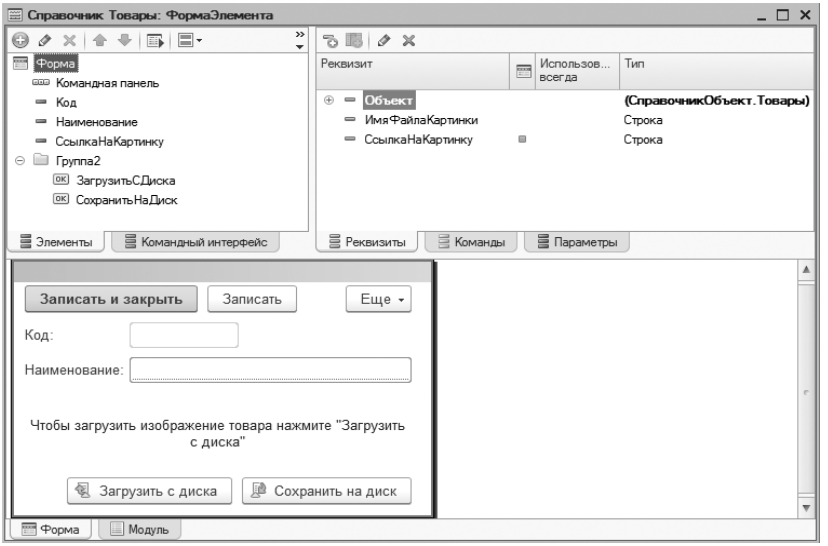


Рис. 3.163. Реквизиты формы справочника «Товары»

Листинг 3.140. Загрузка картинки

```
&НаКлиенте
Процедура ЗагрузитьСДиска(Команда)

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьСДискаЗавершение",
        ЭтотОбъект), , , Истина , УникальныйИдентификатор);

КонецПроцедуры
```

Листинг 3.141. Обработчик оповещения «ЗагрузитьСДискаЗавершение»

```
&НаКлиенте
Процедура ЗагрузитьСДискаЗавершение(Результат, АдресВХранилище, ВыбранноеИмяФайла,
                                    ДополнительныеПараметры) Экспорт

    Если Результат Тогда

        Файл = Новый Файл(ВыбранноеИмяФайла);

        ИмяФайлаКартинки = Файл.Имя;
        ССылкаНаКартинку = АдресВХранилище;

        Модифицированность = Истина;
    КонецЕсли;

КонецПроцедуры
```

Процедуры `НачатьПомещениеФайла()` и `ЗагрузитьСДискаЗавершение()` полностью аналогичны тем, что рассматривалась в листингах 3.130, 3.131.

Как только в реквизит формы мы поместим ссылку на картинку во временном хранилище (`ССылкаНаКартинку = АдресВХранилище`), она отобразится в форме (рис. 3.164).

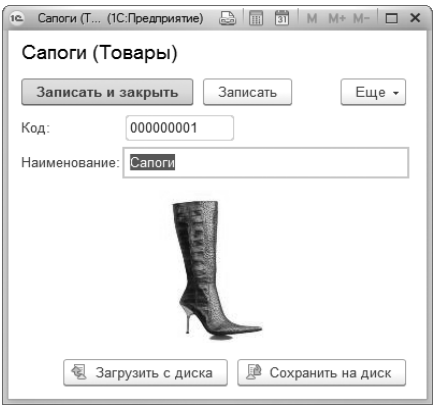


Рис. 3.164. Реквизиты формы справочника «Товары»

Таким образом, после помещения картинки она окажется во временном хранилище, а в реквизите формы будет ссылка на нее. Связанное с реквизитом поле картинки отобразит ее в форме (рис. 3.165).

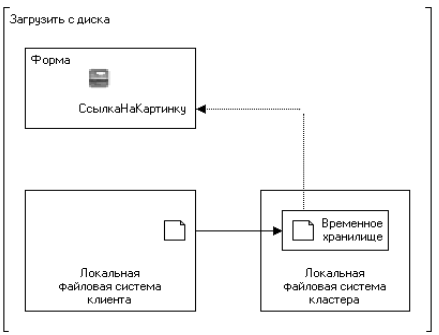


Рис. 3.165. Отображение картинки из временного хранилища

Когда пользователь решит сохранить сделанные изменения, на сервере будет вызван обработчик события формы Перед записью на сервере (листинг 3.142).

Листинг 3.142. Обработчик события «Перед записью на сервере»

```
&НаСервере
Процедура ПередЗаписьюНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    // Получить файл из хранилища и поместить его в объект
    Если ЭтоАдресВременногоХранилища(ССылкаНаКартинку) Тогда
        ДвоичныеДанные = ПолучитьИзВременногоХранилища(ССылкаНаКартинку);
        ТекущийОбъект.ФайлКартинки =
            Новый ХранилищеЗначения(ДвоичныеДанные, Новый СжатиеДанных(9));
        ТекущийОбъект.ИмяФайлаКартинки = ИмяФайлаКартинки;

    КонецЕсли;

КонецПроцедуры
```

Этот обработчик также аналогичен тому, что рассматривался в листинге 3.133. Картинка из временного хранилища помещается в объект справочника, туда же помещается имя картинки из реквизита формы (рис. 3.166).

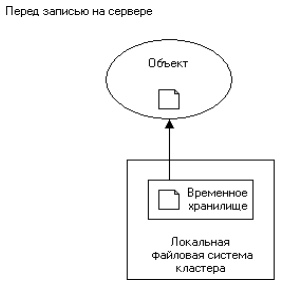


Рис. 3.166. Помещение картинки в реквизит справочника

Затем платформа начнет запись объекта, и его данные будут помещены в базу данных (рис. 3.167).

После этого, еще до окончания транзакции записи, на сервере будет вызван обработчик события формы При записи на сервере (листинг 3.143).

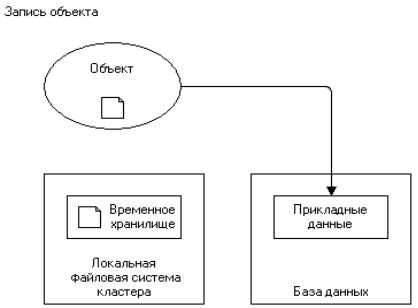


Рис. 3.167. Запись данных справочника в базу данных

Листинг 3.143. Обработчик события «При записи на сервере»

```
&НаСервере
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    // Удалить файл из временного хранилища.
    Если ЭтоАдресВременногоХранилища(ССылкаНаКартинку) Тогда
        УдалитьИзВременногоХранилища(ССылкаНаКартинку);
        ССылкаНаКартинку =
            ПолучитьНавигационнуюССылку(ТекущийОбъект.Ссылка, "ФайлКартинки");

    КонецЕсли;

КонецПроцедуры
```

Здесь есть небольшое отличие от того, что мы делали при работе с файлами.

Как и раньше, мы удаляем картинку из временного хранилища, т. к. она нам больше не нужна, она уже сохранена в базе данных. Но так как форма, как и раньше, должна отображать картинку, в реквизит формы мы помещаем навигационную ссылку на картинку, которая теперь находится в базе данных. Для получения навигационной ссылки указываем ссылку на сам объект и имя его реквизита, в котором хранится картинка (листинг 3.144).

Листинг 3.144. Получение навигационной ссылки на реквизит объекта

```
ССылкаНаКартинку = ПолучитьНавигационнуюССылку(ТекущийОбъект.Ссылка, "ФайлКартинки");
```

В результате мы будем иметь следующую картину (рис. 3.168).

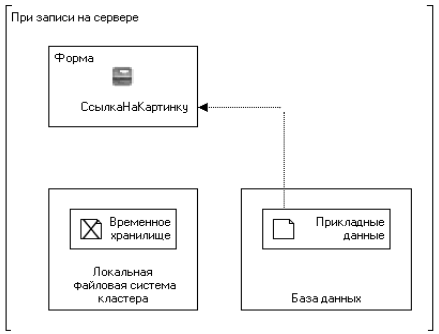


Рис. 3.168. Отображение картинки из реквизита справочника

Теперь осталось сделать последнюю небольшую доработку. Если сейчас закрыть форму и снова открыть сохраненный элемент, мы не увидим картинки в форме, так как в реквизите формы нет ссылки на сохраненную в базе данных картинку. Поэтому создадим обработчик события формы При создании на сервере и в нем поместим в реквизит формы ссылку на картинку (листинг 3.145).

Листинг 3.145. Обработчик события «При создании на сервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

	Если Объект.ИмяФайлаКартинки <> "" Тогда
		СсылкаНаКартинку = ПолучитьНавигационнуюСсылку(Объект.Ссылка, "ФайлКартинки");

	КонецЕсли;

КонецПроцедуры
```

Процесс получения картинки из информационной базы выглядит точно так же, как и в случае работы с файлами (листинг 3.146, рис. 3.169).

Листинг 3.146. Получение картинки из информационной базы

```
&НаКлиенте
Процедура СохранитьНаДиск(Команда)

	Если Объект.ИмяФайлаКартинки = "" Тогда
		ПоказатьПредупреждение(, "У товара нет сохраненной в базе картинки");

	Иначе
```

```
            СсылкаНаФайлВИБ = ПолучитьНавигационнуюССылку(Объект.Ссылка, "ФайлКартинки");
            ПолучитьФайл(СсылкаНаФайлВИБ, Объект.ИмяФайлаКартинки);

        КонецЕсли;

КонецПроцедуры
```

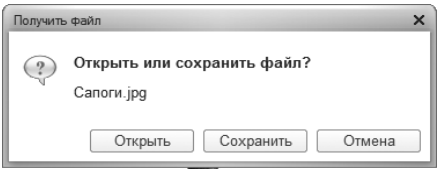


Рис. 3.169. Диалог получения файла

Картинки, используемые для оформления

Кроме картинок, являющихся данными и существующих в виде файлов, в формах могут использоваться картинки, являющиеся элементами оформления. Такие картинки хранятся либо в конфигурации, либо в библиотеке картинок, либо как общие картинки.

Существует три способа использования таких картинок.

- Во-первых, такую картинку можно поместить непосредственно в реквизит формы.
- Во-вторых, картинку, являющуюся коллекцией, можно поместить в свойство элемента формы Поле, и тогда платформа будет отображать одну из имеющихся картинок коллекции. Здесь возможны два варианта:
  - Когда поле связано с реквизитом типа Булево, в коллекции должно быть две картинки: одна из них будет отображаться, когда реквизит имеет значение Истина, другая – когда Ложь.
  - Вторая возможность – связать поле с реквизитом типа Число. Тогда будет отображаться та картинка из коллекции, индекс которой содержится в реквизите.

Рассмотрим перечисленные способы на примере.

Пример можно посмотреть в демонстрационной базе «Файлы и картинки», обработка Формирование маршрутных листов.

В качестве «подопытного кролика» будем использовать форму гипотетической обработки ФормированиеМаршрутныхЛистов.

В ней пользователь должен выбрать сначала тип автомобилей, для которых будут формироваться маршрутные листы: «Грузовики» или «Газели». Чтобы тип выбранных автомобилей был хорошо заметен визуально, в левом углу формы мы выведем картинку, иллюстрирующую тот или иной тип автомобиля.

Чтобы выполнить эту задачу, создадим в конфигурации две общие картинки: Грузовик и Газель (рис. 3.170).

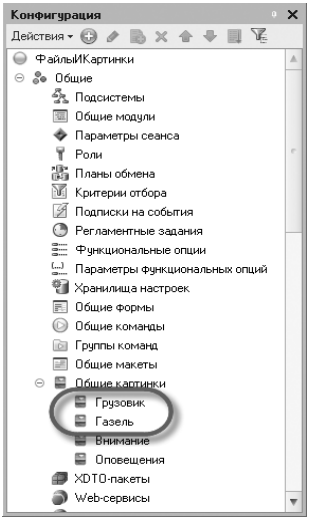


Рис. 3.170. Общие картинки

Затем создадим реквизит формы обработки ТипАвтомобилей типа Число и свяжем его с полем, имеющим вид Поле переключателя (рис. 3.171).

Кроме этого создадим реквизит формы Картинка типа Картинка и свяжем его с полем, имеющим вид Поле картинки (рис. 3.172).

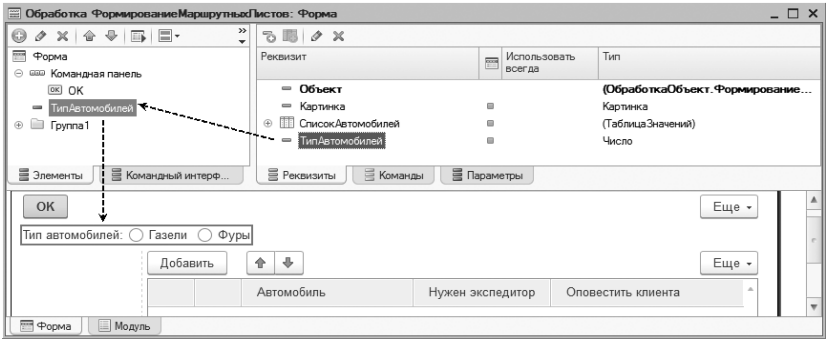


Рис. 3.171. Реквизит формы «ТипАвтомобилей»

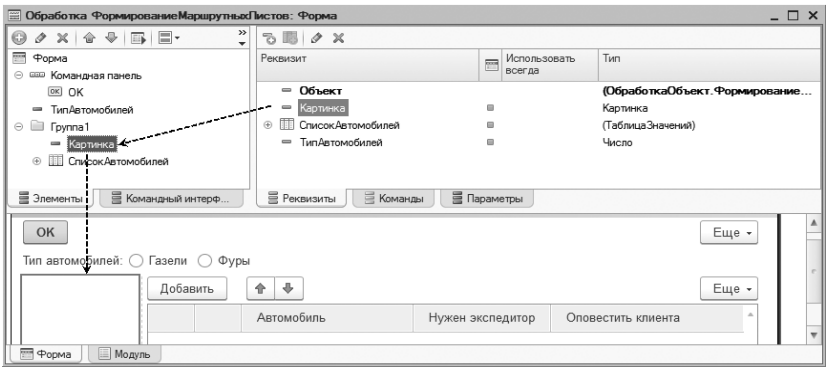


Рис. 3.172. Реквизит формы «Картинка»

Теперь для поля ТипАвтомобилей создадим обработчик события ПриИзменении(), в котором напишем следующий текст (листинг 3.147).

Листинг 3.147. Обработчик события «При изменении»

```
&НаКлиенте
Процедура ТипМашинПриИзменении(Элемент)

    Если ТипАвтомобилей = 1 Тогда
        Картинка = БиблиотекаКартинок.Газель;

    ИначеЕсли ТипАвтомобилей = 2 Тогда
        Картинка = БиблиотекаКартинок.Грузовик;

    КонецЕсли;

КонецПроцедуры
```

В зависимости от выбранного значения переключателя мы подставляем в реквизит формы ту или иную картинку.

Чтобы при открытии формы были выбраны значения переключателя и картинки, в обработчике события формы При создании на сервере напишем такой код (листинг 3.148).

Листинг 3.148. Обработчик события «При создании на сервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    ТипАвтомобилей = 1;
    Картинка = БиблиотекаКартинок.Газель;

КонецПроцедуры
```

В результате после открытия формы она будет иметь следующий вид (рис. 3.173).

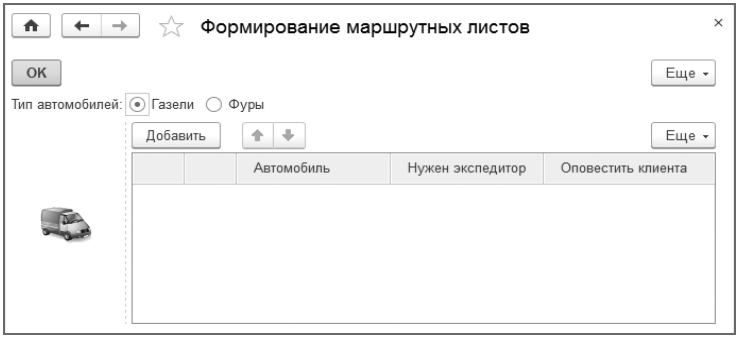


Рис. 3.173. Вид формы после открытия

Если изменить значение переключателя, то изменится и отображаемая в форме картинка (рис. 3.174).

Теперь займемся списком автомобилей.

Кроме идентификатора автомобиля в этом списке указывается, нужен на данный автомобиль экспедитор или нет. Чтобы визуально быстро выделять автомобили, на которые нужен экспедитор, мы будем использовать пиктограмму, выводимую в первой колонке списка.

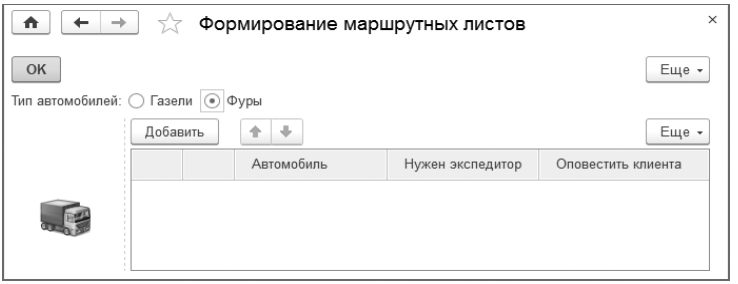


Рис. 3.174. Вид формы после переключения типа автомобилей

Сначала добавим в общие картинки картинку Внимание. Она является коллекцией, состоящей из двух картинок: восклицательного знака и пустой картинки (рис. 3.175).

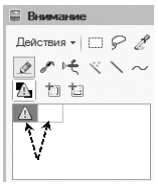


Рис. 3.175. Картинка «Внимание» в режиме коллекции

Если экспедитор требуется, будем показывать восклицательный знак. Если экспедитор не нужен, будем показывать пустую картинку.

Затем создадим реквизит формы СписокАвтомобилей (ТаблицаЗначений) и ее колонки: Автомобиль (Строка) и НуженЭкспедитор (Булево) – рис. 3.176.

Реквизит НуженЭкспедитор свяжем сразу с двумя полями таблицы формы.

Во-первых, с полем СписокАвтомобилейНуженЭкспедитор, имеющим вид Поле флажка. В этом поле мы и будем устанавливать желаемое значение.

Во-вторых, с полем СписокАвтомобилейКартинка, имеющим вид Поле картинки. В этом поле будет отображаться одна из двух пиктограмм.

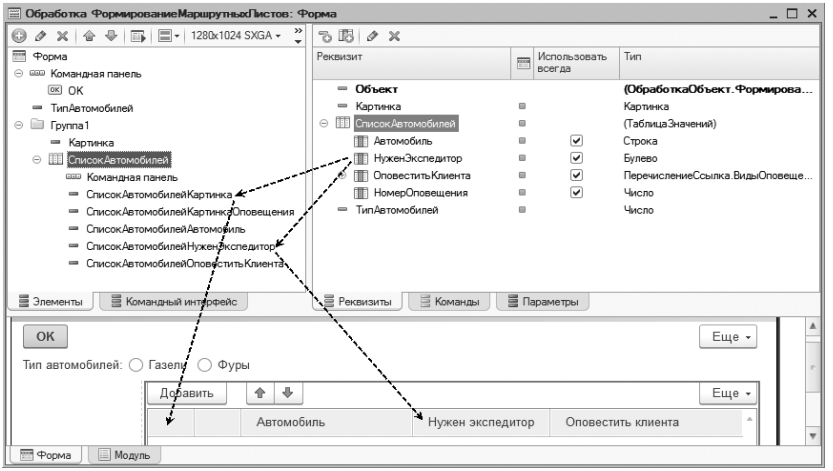


Рис. 3.176. Реквизиты «Автомобиль» и «НуженЭкспедитор»

Чтобы поле СписокАвтомобилейКартинка «знало», где взять пиктограммы, в его свойстве КартинкаЗначений укажем созданную нами общую картинку Внимание (рис. 3.177).

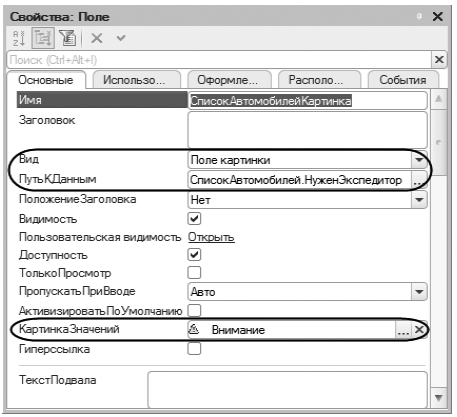


Рис. 3.177. Свойство «Картинка значений»

В результате, если в реквизите НуженЭкспедитор будет значение Истина, отобразится первая картинка из коллекции. Если Ложь – вторая, пустая (рис. 3.178).

Рис. 3.178. Отображение картинки в форме

Теперь рассмотрим последний способ работы с картинками.

В списке автомобилей желательно также указывать, есть ли необходимость оповещать клиента о предстоящей доставке груза, и если есть – указывать конкретный способ оповещения: по телефону, по электронной почте и т. д. При этом хочется видеть визуальные отметки о том или ином виде оповещения.

Для этого добавим в конфигурацию еще одну общую картинку Оповещения. Она также является коллекцией картинок (рис. 3.179).

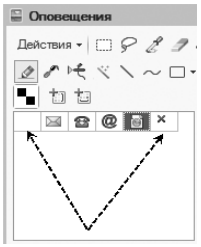


Рис. 3.179. Картинка «Оповещения» в режиме коллекции

Первая картинка в коллекции – пустая, остальные символизируют определенный вид оповещения.

Также добавим в конфигурацию перечисление ВидыОповещений (рис. 3.180).

После этого в таблицу значений в форме добавим два реквизита.

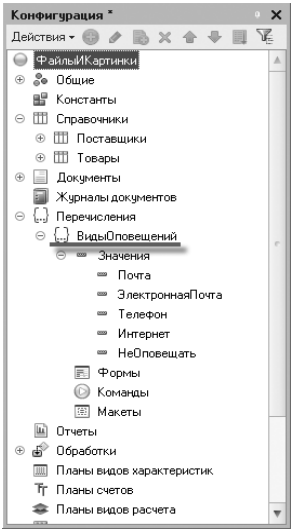


Рис. 3.180. Перечисление «ВидыОповещений»

Первый реквизит – ОповеститьКлиента (с типом ссылки на перечисление ВидыОповещений). Свяжем его с полем таблицы СписокАвтомобилейОповеститьКлиента (рис. 3.181).

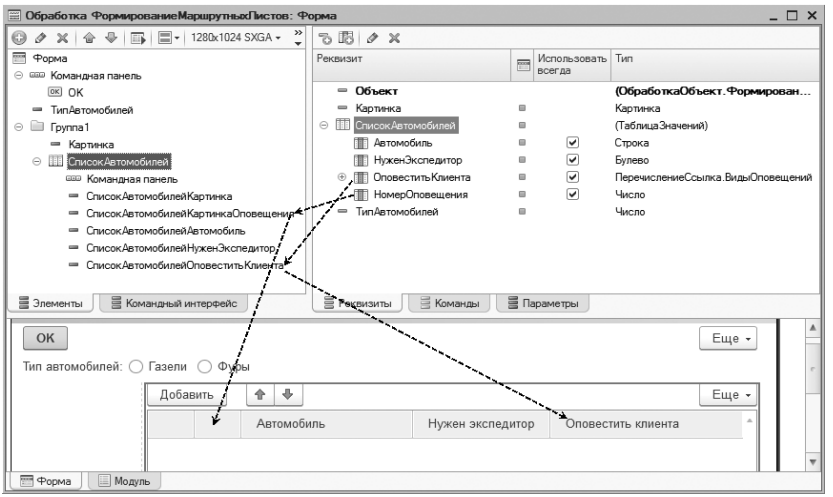


Рис. 3.181. Реквизиты «ОповеститьКлиента» и «НомерОповещения»

В этом поле мы будем устанавливать нужное значение перечисления.

Второй реквизит – НомерОповещения типа Число. Его свяжем с реквизитом таблицы СписокАвтомобилейКартинкаОповещения, имеющим вид Поле картинки. В этом поле будет отображаться одна из выбранных пиктограмм. В его свойстве КартинкаЗначений выберем нашу общую картинку Оповещения.

Для поля таблицы СписокАвтомобилейОповеститьКлиента создадим обработчик события ПриИзменении(), в котором, в зависимости от выбранного значения перечисления, будем заполнять реквизит НомерОповещения (листинг 3.149).

Листинг 3.149. Обработчик события «При изменении»

```
&НаКлиенте
Процедура СписокАвтомобилейОповеститьКлиентаПриИзменении(Элемент)

    ДанныеСтроки = Элементы.СписокАвтомобилей.ТекущиеДанные;
    Если ДанныеСтроки.ОповеститьКлиента =
        ПредопределенноеЗначение("Перечисление.ВидыОповещений.Почта") Тогда
        ДанныеСтроки.НомерОповещения = 1;

    ИначеЕсли ДанныеСтроки.ОповеститьКлиента =
        ПредопределенноеЗначение("Перечисление.ВидыОповещений.Телефон") Тогда
        ДанныеСтроки.НомерОповещения = 2;

    ИначеЕсли ДанныеСтроки.ОповеститьКлиента =
        ПредопределенноеЗначение("Перечисление.ВидыОповещений.ЭлектроннаяПочта") Тогда
        ДанныеСтроки.НомерОповещения = 3;

    ИначеЕсли ДанныеСтроки.ОповеститьКлиента =
        ПредопределенноеЗначение("Перечисление.ВидыОповещений.Интернет") Тогда
        ДанныеСтроки.НомерОповещения = 4;

    ИначеЕсли ДанныеСтроки.ОповеститьКлиента =
        ПредопределенноеЗначение("Перечисление.ВидыОповещений.НеОповещать") Тогда
        ДанныеСтроки.НомерОповещения = 5;

    ИначеЕсли ДанныеСтроки.ОповеститьКлиента.Пустая() Тогда
        ДанныеСтроки.НомерОповещения = 0;

    КонецЕсли;

КонецПроцедуры
```

В результате, если значение перечисления не выбрано, будет отображаться пустая пиктограмма. Если выбрано – пиктограмма, соответствующая этому способу оповещения (рис. 3.182).

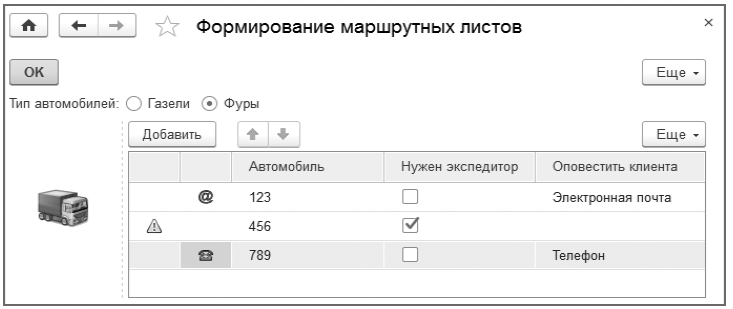


Рис. 3.182. Отображение картинки в форме

Следует сделать небольшое замечание относительно пустых картинок (и картинок вообще), отображаемых в табличной части. Текущая строка списка подсвечивается желтым фоном. Кроме того, к таблице может применяться условное оформление. Поэтому, чтобы пиктограммы хорошо выглядели на любом (не только белом) фоне, нужно делать их фон прозрачным. Особенно это важно для пустых пиктограмм.

В противном случае в текущей строке они будут отображаться в виде белых прямоугольников, что некрасиво и вызывает у пользователя желание нажать на них мышью (рис. 3.183).

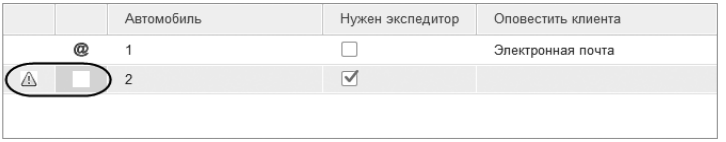


Рис. 3.183. Отображение картинки с непрозрачным фоном

Чтобы сделать пустую картинку прозрачной, нужно отредактировать ее цвет и установить для него свойство Прозрачность в значение `0` (рис. 3.184).

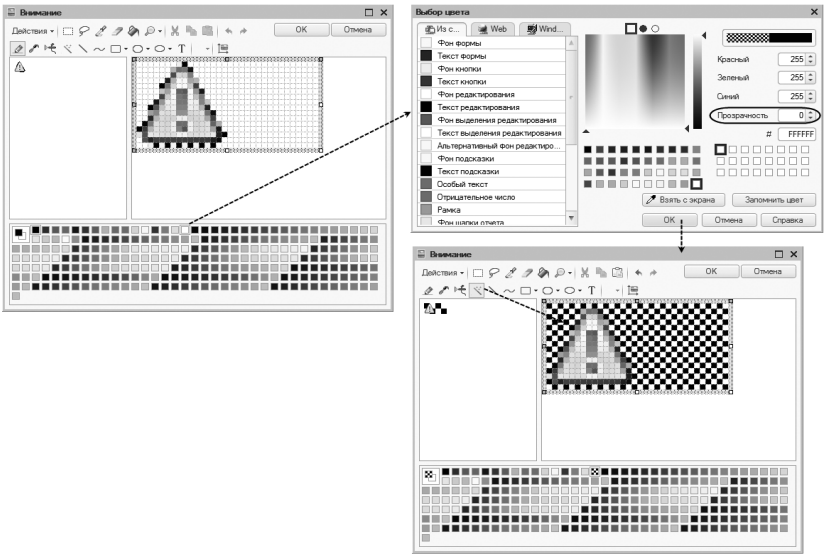


Рис. 3.184. Изменение прозрачности цвета

В результате сделанных изменений проблемы, показанные на рис. 3.183, исчезнут (рис. 3.185).

		Автомобиль	Нужен экспедитор	Оповестить клиента
	@	1	☐	Электронная почта
⚠		2	☑	

Рис. 3.185. Отображение картинки с прозрачным фоном

В информационной базе «Файлы и картинки» у всех картинок уже установлен прозрачный фон.

Глава 3.19. Поле ввода

Для работы с полем ввода ссылочных значений в «1С:Предприятии» существует несколько удобных возможностей:

- ввод значений в поле ввода путем набора текста прямо в этом поле;
- выбор значений из истории выбора в поле ввода;
- создание новых значений при вводе в ссылочное поле.

Ссылочные поля – это поля, значения которых являются ссылками на другие объекты базы данных. Например, в поле Поставщик может содержаться ссылка на какого-либо поставщика, который является одним из элементов справочника Поставщики (рис. 3.186).

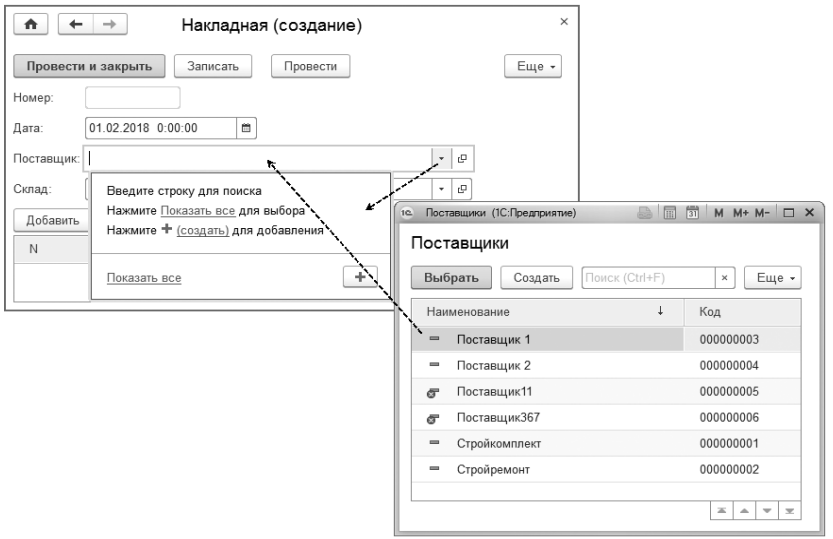


Рис. 3.186. Ссылочное поле «Поставщик»

Ввод по строке

Один из возможных способов ввода такого значения в поле показан на рисунке. Нужно нажать кнопку выпадающего списка в поле ввода, затем в выпадающем списке, появившемся под полем ввода, нажать ссылку Показать все. После этого платформа откроет форму выбора справочника Поставщики, в этой форме необходимо выделить нужного поставщика и нажать кнопку Выбрать. Ссылка на выбранного поставщика будет помещена в поле ввода.

Однако можно использовать другой, более удобный и эффективный способ – *ввод по строке*. Не нажимая кнопку выпадающего списка, просто начать вводить в поле наименование поставщика. В момент паузы или по окончании ввода платформа автоматически сформирует список подходящих поставщиков, из которого можно будет выбрать нужного (рис. 3.187).

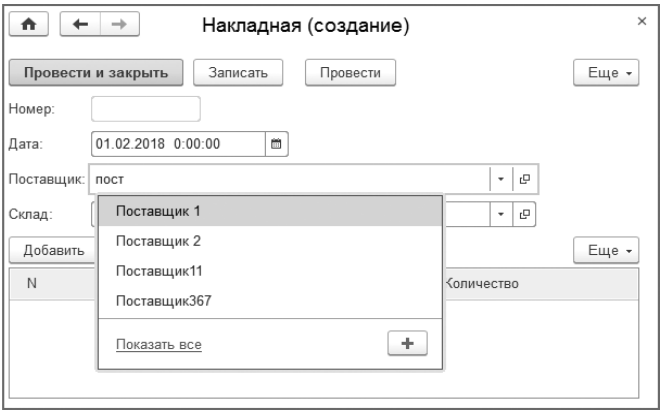


Рис. 3.187. Ввод по строке

Если такой поставщик окажется один, платформа автоматически подставит ссылку на этого поставщика в редактируемое поле при переходе к другому элементу формы.

Ввод по строке возможен благодаря тому, что для каждого объекта конфигурации, имеющего ссылочные значения, автоматически указываются поля, по которым осуществляется ввод по строке (рис. 3.188).

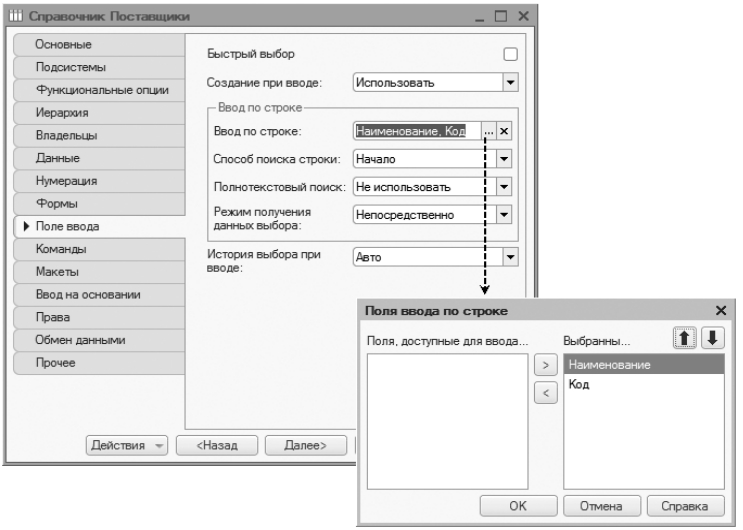


Рис. 3.188. Поля, используемые для ввода по строке

Обычно для справочников в качестве таких полей используются Наименование и Код, то есть можно вводить в поле наименование элемента справочника или можно вводить его код.

Для других объектов конфигурации стандартно могут использоваться другие поля. Разработчик может изменить состав и порядок полей, по которым выполняется ввод по строке.

Последовательность событий при вводе по строке

Рассмотрим, какие события вызываются платформой при вводе по строке (рис. 3.189).

Пользователь начинает вводить текст в поле.

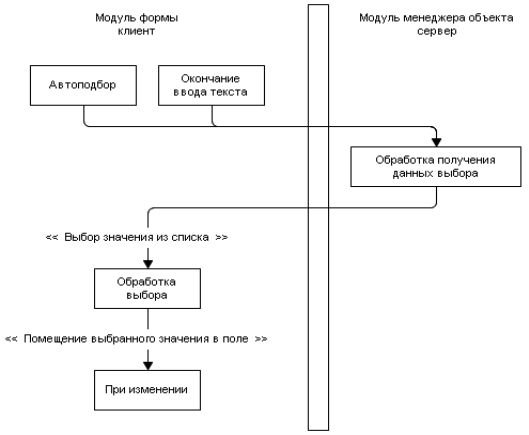


Рис. 3.189. Последовательность событий при вводе по строке

Дальше возможны два варианта развития событий:

- В процессе ввода он делает паузу. В этом случае будет вызвано событие Автоподбор.
- Находясь в поле ввода, в которое уже введена часть наименования, он нажимает стрелку вниз. Например, после паузы платформа показала список выбора, но пользователь случайно закрыл его. В этом случае также будет вызвано событие Автоподбор.
- В процессе ввода он не делает пауз и, введя часть наименования, переходит к следующему элементу формы, нажав Tab или Enter. В этом случае будет вызвано событие Окончание ввода текста (рис. 3.190).

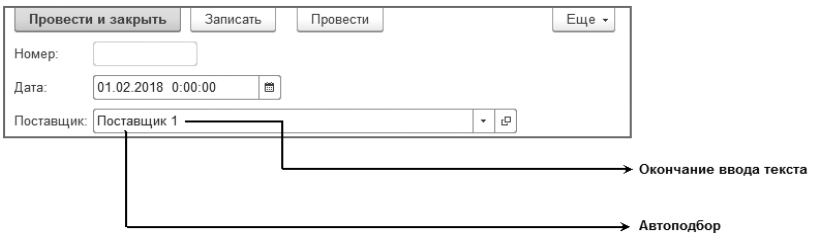


Рис. 3.190. Вызов событий «Автоподбор» и «Окончание ввода текста»

События Автоподбор и Окончание ввода текста – это клиентские события поля формы, вернее, его расширения – расширения поля ввода.

После обработки одного из этих событий платформа вызывает событие Обработка получения данных выбора. Это событие вызывается в модуле менеджера того объекта конфигурации, значение которого хранится в этом поле. В нашем случае это будет событие, обрабатываемое в модуле менеджера справочника Поставщики.

После обработки этого события платформа показывает рядом с полем ввода *список выбора* – список возможных значений, которые могут быть помещены в это поле.

Пользователь выбирает одно из значений, после чего возникает событие поля – Обработка выбора.

После обработки этого события платформа помещает выбранное значение в поле ввода и вызывает последнее событие поля – При изменении.

Чаще всего при работе с вводом по строке разработчик решает две задачи:

- сформировать список выбора по своим правилам;
- отказаться от выбора, сделанного пользователем, или подставить в поле собственное значение.

Вторая задача довольно простая и решается в обработчике Обработка выбора. Подробнее этот вопрос рассматривается в разделе «Событие "Обработка выбора"» на стр. 685.

А вот для решения первой задачи существует большое количество возможностей, которые мы сейчас и рассмотрим.

Формирование собственного списка выбора

Как видно из приведенной выше схемы, в любом случае при вводе по строке вызываются два события:

- Автоподбор или Окончание ввода текста;
- Обработка получения данных выбора.

Примечательным здесь является то, что первые события вызываются в форме, в то время как последнее событие вызывается в модуле менеджера прикладного объекта.

Из этого следует важное замечание: переопределять формирование списка выбора нужно прежде всего в обработчике Обработка получения данных выбора. Потому что это будет работать во всех случаях, когда в каком-либо поле, в какой-либо форме будет формироваться список выбора значений этого типа. Даже в тех формах, которые генерируются платформой автоматически и которые разработчик изменить не может.

Более сложный случай – когда в одной или нескольких формах нужно иметь особенное формирование списка выбора, не такое, как в остальных местах. В этом случае нужно использовать обработчики событий этой (этих) формы: Автоподбор и Окончание ввода текста. В простейшем варианте в этих обработчиках должен быть написан одинаковый алгоритм.

Самым сложным и специфическим является случай, когда в некоторой форме список выбора при паузе и при переходе к следующему элементу должен формироваться по разным алгоритмам. Тогда в каждом из обработчиков Автоподбор и Окончание ввода текста будет собственный алгоритм формирования списка выбора.

Событие «Обработка получения данных выбора»

Итак, рассмотрим возможности, которые существуют в обработчике события Обработка получения данных выбора.

Синтаксис описания этого обработчика выглядит следующим образом (листинг 3.150).

Листинг 3.150. Объявление обработчика события «Обработка получения данных выбора»

```
ОбработкаПолученияДанныхВыбора(ДанныеВыбора, Параметры, СтандартнаяОбработка)
```

ДанныеВыбора – это переменная, в которую разработчик должен поместить собственный список выбора. При входе в обработчик этот параметр не содержит никаких значений, то есть через него нельзя получить доступ к списку выбора, который сформирует платформа. Просто потому, что этот список формируется уже после выхода

из этого обработчика. Однако может возникнуть желание в этом обработчике получить список, формируемый платформой, и добавить в него (удалить) несколько элементов. Как это сделать – рассказано в разделе «Метод "ПолучитьДанныеВыбора()"» на стр. 682.

Переменная Параметры содержит набор параметров, которые платформа будет использовать для формирования списка выбора. Особенность заключается в том, что при формировании списка выбора платформа будет учитывать свойства Параметры выбора и Связи параметров выбора, заданные для соответствующего реквизита объекта конфигурации. Поэтому в параметрах могут содержаться какие-то отборы.

СтандартнаяОбработка – это булева переменная, на основе которой платформа определяет, что делать после выхода из этого обработчика.

Если СтандартнаяОбработка равна Истина, то платформа самостоятельно сформирует список выбора исходя из того, что указано в Параметрах.

Если СтандартнаяОбработка равна Ложь, платформа не будет формировать список выбора самостоятельно, а покажет то, что находится в параметре ДанныеВыбора.

Таким образом, у нас есть следующие возможности:

- изменить Параметры и сказать платформе, чтобы она сформировала список выбора;
- отказаться от стандартной обработки и самостоятельно сформировать список выбора.

Сначала рассмотрим, каким образом можно модифицировать параметры.

Пример можно посмотреть в демонстрационной базе «Поле ввода», документ Накладная, реквизит Товар табличной части.

Допустим, есть документ Накладная, в табличной части которого необходимо подбирать товары. Причем для выбора должны предлагаться не все возможные товары, а только те, которые поставляет поставщик, указанный в этом документе, и те, которые не помечены на удаление.

Для этого у реквизита табличной части Товар заданы свойства Связи параметров выбора и Параметры выбора (рис. 3.191).

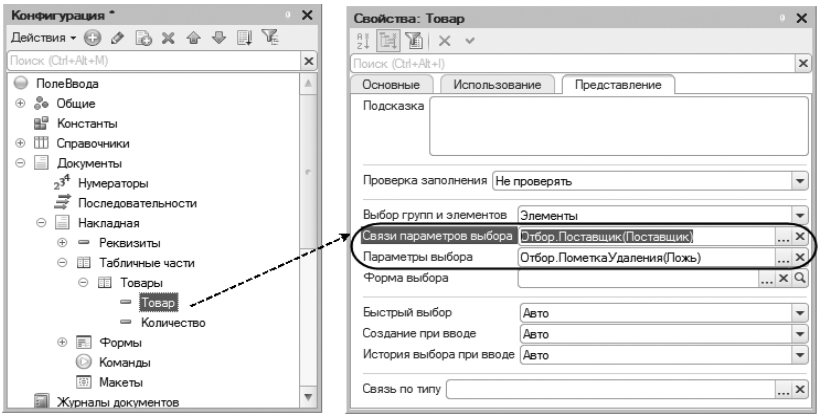


Рис. 3.191. Свойства «Связи параметров выбора» и «Параметры выбора»

В связях параметров выбора указывается, что для выбора будут предлагаться только товары поставщика, указанного в документе. В параметрах выбора указывается, что кроме этого для выбора будут предлагаться только товары, не помеченные на удаление.

В результате при наборе наименования товара в форме будет предложен следующий список товаров (рис. 3.192).

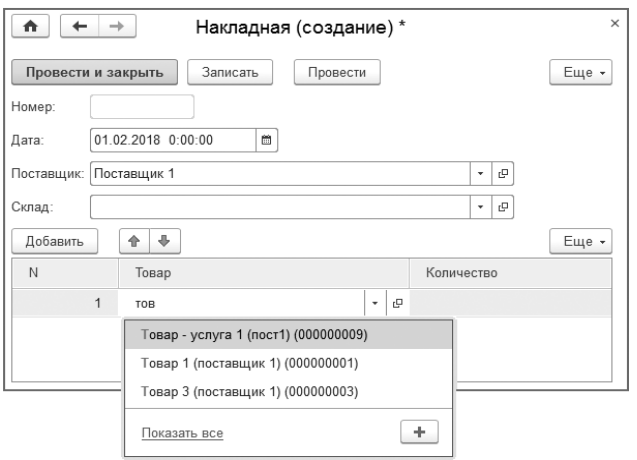


Рис. 3.192. Товары от поставщика «Поставщик 1», не помеченные на удаление

Если мы посмотрим на все имеющиеся товары, то заметим, что подбор товаров произведен правильно (рис. 3.193).

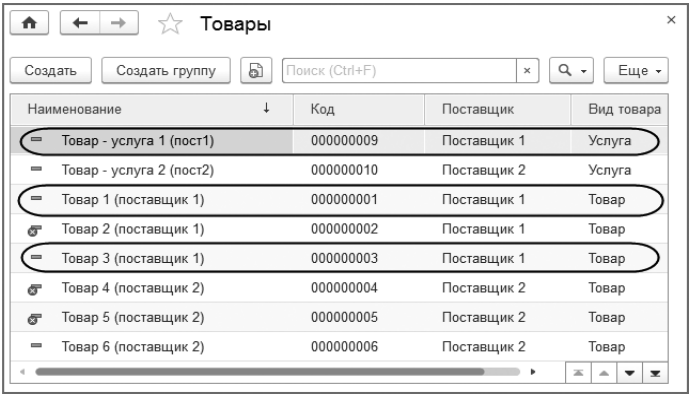


Рис. 3.193. Полный список товаров

Допустим, нас не устраивает, что для выбора предлагаются те элементы справочника, которые являются услугами.

Мы можем легко исправить это сразу двумя способами. Можно просто, ничего не программируя, добавить этот отбор в свойство Параметры выбора реквизита табличной части Товар.

Но, если нужно, можно сделать это программно в обработчике события Обработка получения данных выбора в модуле менеджера справочника Товары.

Благодаря свойствам Параметры выбора и Связи параметров выбора в этом обработчике структура параметров содержит следующие отборы (рис. 3.194).

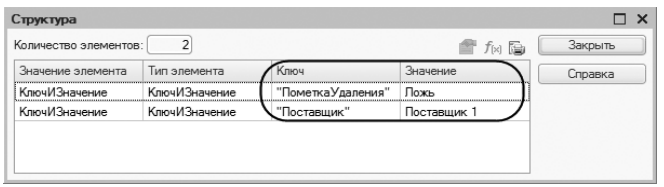


Рис. 3.194. Структура отборов

Добавим к ним еще одно условие – что товар не должен быть услугой, а стандартную обработку оставим в значении Истина, чтобы платформа сформировала список выбора самостоятельно (листинг 3.151).

Листинг 3.151. Добавление условия отбора

```
Процедура ОбработкаПолученияДанныхВыбора(ДанныеВыбора, Параметры, СтандартнаяОбработка)

    Параметры.Отбор.Вставить("ВидТовара", Перечисления.ВидыТоваров.Товар);

КонецПроцедуры
```

Если теперь выполнить те же самые операции, мы увидим, что для выбора предлагаются только элементы, являющиеся товарами (рис. 3.195).

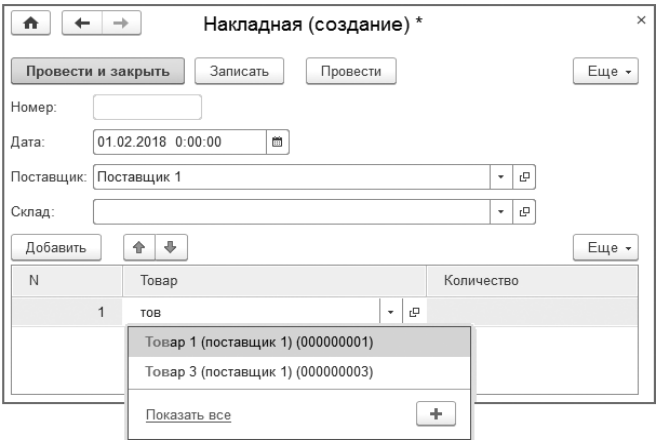


Рис. 3.195. Товары, предлагаемые для выбора

Рассмотрим вторую ситуацию, когда мы полностью самостоятельно формируем список выбора, не используя стандартные возможности платформы.

Здесь есть два случая. Список выбора может быть «простым» и «сложным».

- В простом случае это должен быть список значений, содержащий выбираемые значения (ссылки).

- В сложном случае это будет список значений, содержащий в качестве своих элементов структуры. Каждая такая структура содержит само выбираемое значение и некоторую дополнительную информацию об этом значении.

Пример можно посмотреть в демонстрационной базе «Поле ввода», документ Накладная, реквизит Склад.

Сначала посмотрим, как можно сформировать простой список.

Допустим, в документе Накладная есть реквизит Склад. Тогда в модуле менеджера справочника Склады мы можем написать следующий обработчик (листинг 3.152).

Листинг 3.152. Обработчик события «Обработка получения данных выбора»

```
Процедура ОбработкаПолученияДанныхВыбора(ДанныеВыбора, Параметры, СтандартнаяОбработка)

	СтандартнаяОбработка = Ложь;

	Запрос = Новый Запрос;
	Запрос.Текст =
		"ВЫБРАТЬ
		|	Склады.Ссылка
		|ИЗ
		|	Справочник.Склады КАК Склады
		|ГДЕ
		|	Склады.Розничный = ЛОЖЬ
		|	И Склады.Наименование ПОДОБНО &СтрокаПоиска";

	Запрос.УстановитьПараметр("СтрокаПоиска", "%" + Параметры.СтрокаПоиска + "%");
	Результат = Запрос.Выполнить();

	ВыборкаДетальныеЗаписи = Результат.Выбрать();

	Список = Новый СписокЗначений;
	Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
		Список.Добавить(ВыборкаДетальныеЗаписи.Ссылка);

	КонецЦикла;

	ДанныеВыбора = Список;

КонецПроцедуры
```

В нем мы прежде всего отказываемся от стандартной обработки.

Затем формируем запрос, который выберет нам все склады, не являющиеся розничными и наименование которых содержит строку, введенную пользователем в поле ввода.

Строку, введенную пользователем, мы получаем из структуры параметров, в ней она содержится отдельным элементом с ключом СтрокаПоиска (листинг 3.153).

Листинг 3.153. Получение строки, введенной пользователем

```
Запрос.УстановитьПараметр("СтрокаПоиска", "%" + Параметры.СтрокаПоиска + "%");
```

Затем создаем пустой список значений и, обходя результат запроса, заполняем список ссылками. В конце помещаем этот список в переменную ДанныеВыбора.

В результате при вводе наименования склада мы будем иметь следующую ситуацию (рис. 3.196).

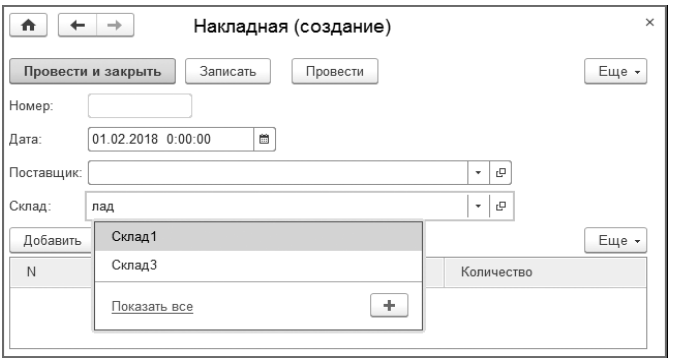


Рис. 3.196. Склады, предлагаемые для выбора

Если посмотреть на список складов, мы увидим, что выбор произведен правильно, а фрагмент наименования склада может быть произвольным, не обязательно находящимся в начале наименования (рис. 3.197).

Теперь рассмотрим, как формировать сложный список выбора.

Допустим, в документе Накладная есть реквизит Поставщик. Тогда в модуле менеджера справочника Поставщики мы можем написать следующий обработчик (листинг 3.154).

Пример можно посмотреть в демонстрационной базе «Поле ввода», документ Накладная, реквизит Поставщик. В модуле менеджера справочника Поставщики снять комментарии с варианта «Пример 1».

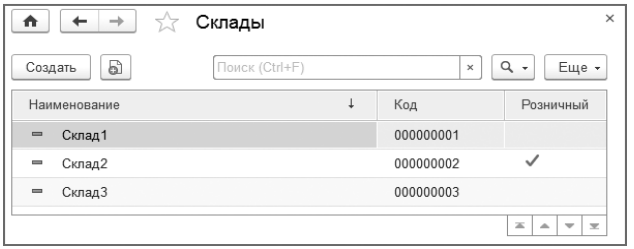


Рис. 3.197. Полный список складов

Листинг 3.154. Обработчик события «Обработка получения данных выбора»

```
Процедура ОбработкаПолученияДанныхВыбора(ДанныеВыбора, Параметры, СтандартнаяОбработка)

	СтандартнаяОбработка = Ложь;

	Запрос = Новый Запрос;
	Запрос.Текст =
		"ВЫБРАТЬ
		|	Поставщики.Ссылка,
		|	Поставщики.ПометкаУдаления,
		|	Поставщики.Ненадежный
		|ИЗ
		|	Справочник.Поставщики КАК Поставщики
		|ГДЕ
		|	Поставщики.Наименование ПОДОБНО &СтрокаПоиска";

	Запрос.УстановитьПараметр("СтрокаПоиска", "%" + Параметры.СтрокаПоиска + "%");
	Результат = Запрос.Выполнить();
	ВыборкаДетальныеЗаписи = Результат.Выбрать();

	Список = Новый СписокЗначений;
	Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
		Структура = Новый Структура;
		Структура.Вставить("Значение", ВыборкаДетальныеЗаписи.Ссылка);
		Структура.Вставить("ПометкаУдаления", ВыборкаДетальныеЗаписи.ПометкаУдаления);

		Если ВыборкаДетальныеЗаписи.Ненадежный Тогда
			Структура.Вставить("Предупреждение", "Это ненадежный поставщик,
				его лучше не выбирать!");

		КонецЕсли;

		Список.Добавить(Структура);

	КонецЦикла;

	ДанныеВыбора = Список;

КонецПроцедуры
```

Здесь, как и раньше, отказываемся от стандартной обработки и запросом получаем нужные ссылки.

Затем создаем пустой список значений, а при каждом обходе результата запроса создаем структуру, в которую помещаем:

- Само значение – в элемент с ключом `Значение`.
- Признак пометки удаления – в элемент с ключом `ПометкаУдаления`. Если пользователь выберет поставщика, помеченного на удаление, платформа выдаст стандартное предупреждение.
- Для тех поставщиков, которые являются ненадежными, мы добавляем в структуру элемент с ключом `Предупреждение`. Если пользователь выберет ненадежного поставщика, ему будет выдано добавленное нами предупреждение.

После заполнения структуры добавляем ее в список значений, а после обхода всего результата запроса список помещаем в переменную `ДанныеВыбора`.

В результате при вводе наименования поставщика список выбора будет выглядеть следующим образом (рис. 3.198).

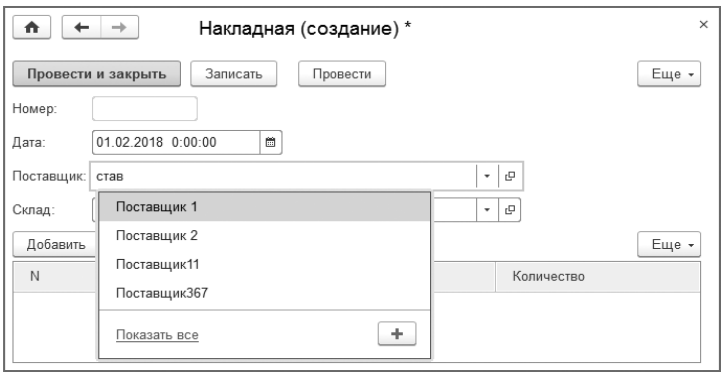


Рис. 3.198. Поставщики, предлагаемые для выбора

На примере имеющихся поставщиков мы сможем посмотреть все возможные варианты работы «сложного» списка выбора (рис. 3.199).

Если выбрать `Поставщик 1`, мы не получим никаких предупреждений.

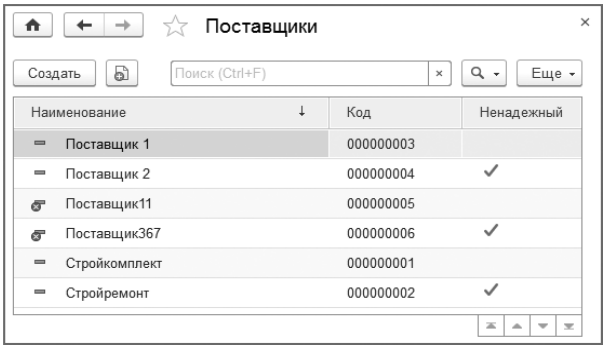


Рис. 3.199. Полный список поставщиков

Если выбрать поставщика, помеченного на удаление (Поставщик11), мы получим стандартное предупреждение платформы (рис. 3.200).

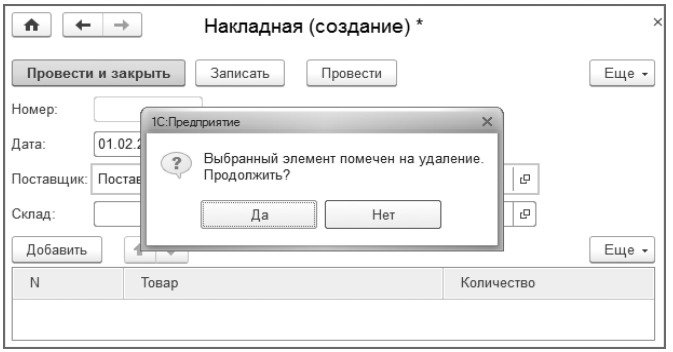


Рис. 3.200. Предупреждение о том, что выбранный элемент помечен на удаление

Если выбрать ненадежного поставщика, да еще и помеченного на удаление (Поставщик367), мы получим «свое» предупреждение (рис. 3.201).

Таким образом, попутно мы убедились в следующем: если для выбираемого элемента указаны и пометка удаления, и предупреждение, будет выведено предупреждение.

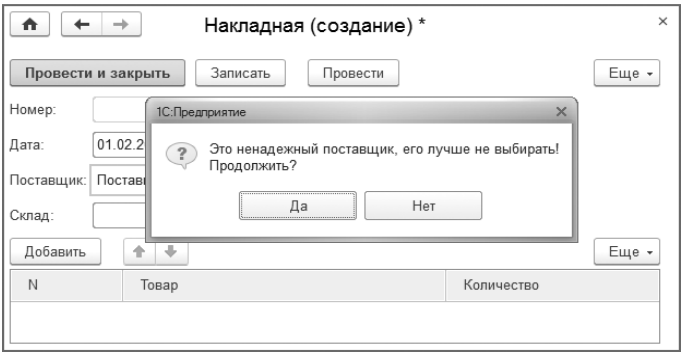


Рис. 3.201. Предупреждение о том, что выбран ненадежный поставщик

Несколько слов о производительности. Событие Обработка получения данных выбора, обрабатываемое в модуле менеджера объекта, вызывается довольно часто. Перечислим эти случаи еще раз:

- Во-первых, при автоподборе, то есть когда возникает пауза при наборе текста в поле ввода.
- Во-вторых, при вводе по строке, то есть когда в поле ввода уже введена какая-то строка и фокус ввода переходит на другой элемент формы.
- И в-третьих (о чем не упоминалось в этом разделе), при быстром выборе, когда для выбора объекта используется не его основная форма выбора, а небольшой список, формируемый системой.

Поэтому при написании обработчика Обработка получения данных выбора нужно особенно тщательно подходить к вопросу эффективности создаваемого кода.

Метод «ПолучитьДанныеВыбора()»

В процессе работы с полем ввода могут возникнуть самые разные ситуации, связанные с обработкой ввода по строке или автоподбора.

Как мы уже говорили, ввод по строке возможен только для ссылочных значений. Однако автоподбор и окончание ввода текста отрабатываются для любых типов в поле ввода.

Поэтому, например, поле ввода может иметь строковый тип, но в обработчике события Автоподбор может анализироваться один или несколько справочников, из которых будет получаться список элементов, соответствующих введенной подстроке.

Другой случай, когда поле ввода имеет ссылочный тип и при обработке события Обработка получения данных выбора в модуле менеджера объекта еще до того, как платформа сформирует список выбора, необходимо получить этот список для модификации.

Для таких особенных случаев у менеджеров прикладных объектов конфигурации существуют методы ПолучитьДанныеВыбора(). Эти методы полностью имитируют формирование списка выбора платформой, вплоть до вызова события Обработка получения данных выбора.

Такой метод есть и в глобальном контексте. В его параметре нужно указывать тип объекта, для которого он должен быть вызван.

В использовании этих методов нет ничего сложного, за исключением того случая, когда этот метод вызывается в обработчике Обработка получения данных выбора. Если не предпринять специальных действий, это приведет к бесконечной рекурсии. Но от нее можно избавиться довольно простым способом. Рассмотрим это на примере.

Пример можно посмотреть в демонстрационной базе «Поле ввода», документ Накладная, реквизит Поставщик. В модуле менеджера справочника Поставщики снять комментарии с варианта «Пример 2».

Рассмотрим следующий обработчик в модуле менеджера справочника Поставщики (листинг 3.155).

Листинг 3.155. Обработчик события «Обработка получения данных выбора»

```
Процедура ОбработкаПолученияДанныхВыбора(ДанныеВыбора, Параметры, СтандартнаяОбработка)

    // Если это рекурсивный вызов – ничего не делать.
    Если НЕ Параметры.Свойство("Рекурсия") Тогда
        СтандартнаяОбработка = Ложь;

        // Получить стандартный список выбора.
        Параметры.Вставить("Рекурсия");
        СтандартныйСписок = ПолучитьДанныеВыбора(Параметры);
```

```
            // Добавить к стандартному списку собственный элемент.
            ДополнительныйЭлемент = Справочники.Поставщики.НайтиПоКоду("000000003");
            СтандартныйСписок.Добавить(ДополнительныйЭлемент);

            ДанныеВыбора = СтандартныйСписок;

        КонецЕсли;

КонецПроцедуры
```

Для защиты от рекурсии перед вызовом метода ПолучитьДанныеВыбора() в структуру параметров будем добавлять служебный параметр с ключом Рекурсия.

Весь код, написанный в обработчике, будем выполнять только в том случае, если в параметрах отсутствует элемент Рекурсия, т. е. при первом «заходе» в этот обработчик (листинг 3.156).

Листинг 3.156. Проверка на отсутствие рекурсии

```
Если НЕ Параметры.Свойство("Рекурсия") Тогда
```

При втором заходе в этот обработчик (когда платформа будет формировать стандартный список выбора) в параметрах уже будет присутствовать элемент Рекурсия. Таким образом, никакой код выполняться не будет (листинг 3.157).

Листинг 3.157. Добавление элемента «Рекурсия» в данные выбора

```
Параметры.Вставить("Рекурсия");
СтандартныйСписок = ПолучитьДанныеВыбора(Параметры);
```

После получения стандартного списка мы добавляем в него еще одного поставщика, который нам очень нравится и, возможно, пригодится пользователю для выбора (листинг 3.158).

Листинг 3.158. Добавление «собственного» поставщика

```
ДополнительныйЭлемент = Справочники.Поставщики.НайтиПоКоду("000000003");
СтандартныйСписок.Добавить(ДополнительныйЭлемент);
```

И в конце полученный список помещаем в переменную ДанныеВыбора. Так как в начале мы отказались от стандартной обработки, именно этот список и будет показан пользователю (рис. 3.202).

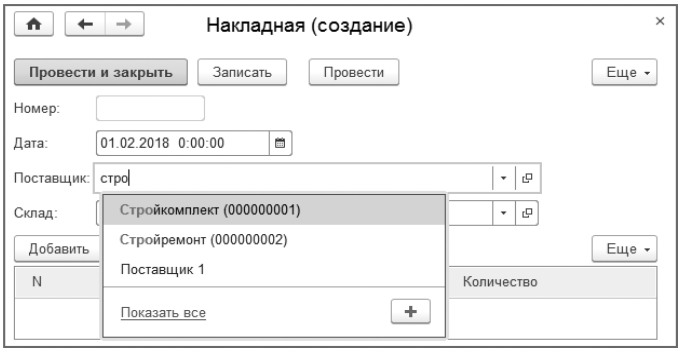


Рис. 3.202. Список поставщиков, предложенных для выбора

Событие «Обработка выбора»

Это событие хорошо тем, что оно вызывается после того, как пользователь выбрал какое-то значение из предложенного списка, но еще до того, как это значение будет помещено в поле ввода.

Поэтому в этом событии можно:

- отказаться от того выбора, который сделал пользователь (`СтандартнаяОбработка = Ложь`);
- поместить в поле ввода другое значение, не то, которое выбрал пользователь (параметр `ВыбранноеЗначение`).

Первая ситуация может показаться на первый взгляд странной. Если пользователь может выбрать «что-то не то», то зачем тогда предлагать ему это выбирать? Нужно с самого начала предложить ему выбирать только из того, что «можно».

Такой подход имеет право на жизнь и, наверное, может применяться в целом ряде случаев. Однако он имеет два недостатка.

Первый недостаток, если можно так выразиться, «юзабильный». Например, пользователь начинает вводить в документ поставщика Стройтрест. Он точно знает, что такой поставщик существует в базе данных. Но программа не предлагает ему выбрать этого поставщика. Первое, о чем подумает пользователь, – что программа «сломалась» или что он неправильно пишет наименование. О том, что именно этого поставщика нельзя выбрать именно в этом документе, пользова-

тель может не подумать совсем или догадается об этом только после долгих экспериментов.

Поэтому логичнее поступать следующим образом: если пользователь хочет выбрать поставщика с таким наименованием, предоставить ему такую возможность. Но если по каким-то причинам этого поставщика нельзя использовать в этом документе, после выбора пользователя сообщить ему об этом и отказаться от выбора, сделанного пользователем.

Такой сценарий, конечно, не является догмой, но в целом ряде случаев, когда для пользователя не очевиден тот факт, что выбирать можно среди ограниченного множества элементов, этот сценарий вполне имеет право на жизнь.

Другой недостаток изначального ограничения списка выбора может быть чисто технологическим. Например, для того чтобы выяснить, можно использовать этого поставщика или нет, требуется выполнить целый ряд сложных и непростых расчетов.

Допустим, список выбора, формируемый для пользователя, будет содержать 49 элементов. Для всех них необходимо будет выполнить этот расчет. Это может занять значительное время. А в результате пользователь выберет один-единственный элемент из этих 49.

В такой ситуации гораздо производительнее будет, ничего не рассчитывая, предоставить ему для выбора 49 элементов. А после того как он выберет единственный элемент, только для него одного провести необходимые расчеты. И либо согласиться с выбором пользователя, либо отказаться от него.

Что касается помещения в поле ввода значения другого типа, такая ситуация может возникнуть при имитации ввода по строке, выполняемого платформой. Например, когда поле ввода имеет строковый тип, список выбора формируется из элементов справочника, на основании выбора пользователя заполняется какая-то служебная информация, а в поле ввода помещается не ссылка, выбранная пользователем, а ее представление.

Выполнить все эти действия как раз и можно в обработчике события Обработка выбора.

Событие «Начало выбора»

Для того чтобы картина с формированием списка выбора для поля ввода была полной, следует упомянуть еще об одном событии – Начало выбора.

Это событие возникает в поле ввода в момент нажатия кнопки выбора или клавиши F4 либо вызывается при нажатии Показать все или по клавише F4 в выпадающем списке поля ввода (рис. 3.203).

Рис. 3.203. Событие «Начало выбора»

Стандартным действием платформы в этом случае является либо открытие формы выбора, либо открытие списка выбора (если для реквизита указан режим быстрого выбора).

Если разработчик переопределяет формирование списка выбора при автоподборе и при окончании ввода текста, вполне возможно, что ему захочется формировать подобный список и при начале выбора.

Такая возможность существует. Обработчик события Начало выбора имеет параметры `ДанныеВыбора` и `СтандартнаяОбработка`. Они используются таким же образом, как и в других рассмотренных обработчиках (листинг 3.159).

Листинг 3.159. Объявление обработчика события «Начало выбора»

```
&НаКлиенте
Процедура ПоставщикНачалоВыбора(Элемент, ДанныеВыбора, СтандартнаяОбработка)
```

Можно отменить стандартную обработку, а в данные выбора поместить список значений, из которых будет выбирать пользователь. Тогда, даже если для реквизита не используется быстрый выбор, при наступлении события будет открыта не форма выбора, а список выбора, расположенный под полем ввода.

История выбора при вводе

При вводе значений в ссылочное поле также очень удобно пользоваться выпадающим списком, открывающимся под этим полем, содержащим наиболее часто используемые и последние выбранные значения данного ссылочного типа.

Использование истории выбора задается с помощью свойства ИсторияВыбораПриВводе у прикладных объектов конфигурации (справочников, документов, перечислений и т. п.), ссылочных реквизитов объектов конфигурации, измерений, ресурсов и др. и элементов формы, отображающих данные этих реквизитов.

Если на всех «уровнях» иерархии (начиная от поля ввода до прикладного объекта) свойство ИсторияВыбораПриВводе установлено в значение Авто, то история выбора при вводе в ссылочное поле будет сохраняться и отображаться в том случае, если это поле не находится в режиме выбора из списка или в режиме быстрого выбора (рис. 3.204).

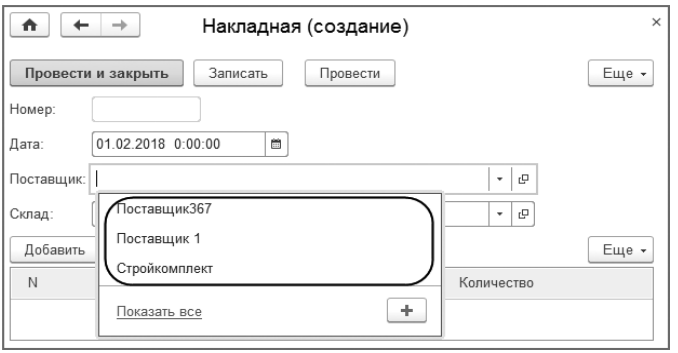


Рис. 3.204. История выбора при вводе ссылочных значений

Выпадающий список с историей выбора открывается автоматически при любом попадании фокуса в пустое ссылочное поле ввода, или если список подбора значений при вводе по строке пустой.

Однако в некоторых ситуациях пользователю может быть совершенно не нужно (и даже вредно) видеть значения, которые он выбирал в поле ввода до этого. Например, когда необходимо выбирать и закрывать

заказы покупателей. Естественно, в этом случае он должен видеть только те заказы, которые он не обрабатывал ранее.

В этом случае нужно запретить показ истории выбора при вводе либо на уровне реквизита объекта конфигурации, либо для отдельного поля ввода. Для этого нужно установить у него свойство ИсторияВыбораПриВводе в значение Не использовать.

Это можно сделать как интерактивно, так и программно, в зависимости от программной логики работы. Например, при выполнении определенных условий показ и формирование истории выбора у поля ввода можно включать или выключать с помощью встроенного языка (листинг 3.160).

Листинг 3.160. Включение/отключение истории выбора у поля ввода

```
Если ... Тогда
        Элементы.Склад.ИсторияВыбораПриВводе = ИсторияВыбораПриВводе.НеИспользовать;
Иначе
        Элементы.Склад.ИсторияВыбораПриВводе = ИсторияВыбораПриВводе.Авто;
КонецЕсли;
```

Кроме того, отключать историю выбора следует для объектов, в модуле менеджера которых переопределена обработка получения данных выбора (есть обработчик ОбработкаПолученияДанныхВыбора), т. к. прописанные там условия не учитываются механизмом составления списка истории выбора. Поэтому, используя историю выбора в этом случае, пользователь может получить возможность выбрать значение, которое он не мог бы выбрать другими способами.

Создание при вводе

Еще одной удобной возможностью при вводе данных в ссылочное поле, несомненно, является возможность создания нового элемента ссылочного типа непосредственно из выпадающего списка, открывающегося под этим полем.

Эта возможность задается с помощью свойства СозданиеПриВводе у прикладных объектов конфигурации (справочников, документов и т. п.), ссылочных реквизитов объектов конфигурации, измерений,

ресурсов и др. У элементов формы, отображающих данные ссылочных реквизитов, это свойство называется КнопкаСоздания.

Если на всех «уровнях» иерархии (начиная от поля ввода до прикладного объекта) создание новых элементов данных при вводе в ссылочное поле разрешено, то в выпадающем списке у поля будет присутствовать кнопка создания – с пиктограммой «+». С ее помощью пользователь может создавать новые элементы данных в форме прикладного объекта и сразу же подставлять их в поле ввода, не открывая при этом форму выбора (рис. 3.205).

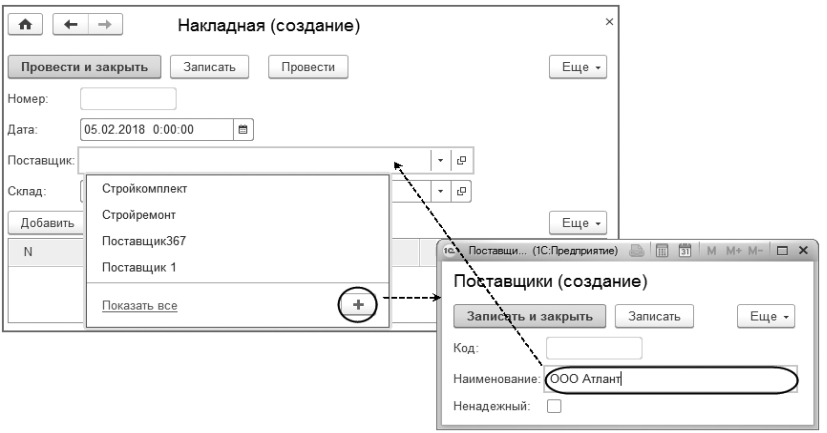


Рис. 3.205. Создание нового элемента справочника при вводе в ссылочное поле

Например, пользователь может сразу начинать ввод наименования нового поставщика в ссылочном поле Поставщик. Убедившись, что соответствия с новым значением не найдено, он может нажать кнопку создания в выпадающем списке, открывшемся под этим полем. Появится форма нового элемента справочника Поставщики, при этом введенные в поле ввода символы уже будут подставлены в наименование нового элемента. Пользователю останется только нажать кнопку Записать и закрыть, и ссылка на нового поставщика будет подставлена в поле ввода (рис. 3.206).

Таким образом существенно повышаются комфортность и скорость работы пользователей с прикладным решением.

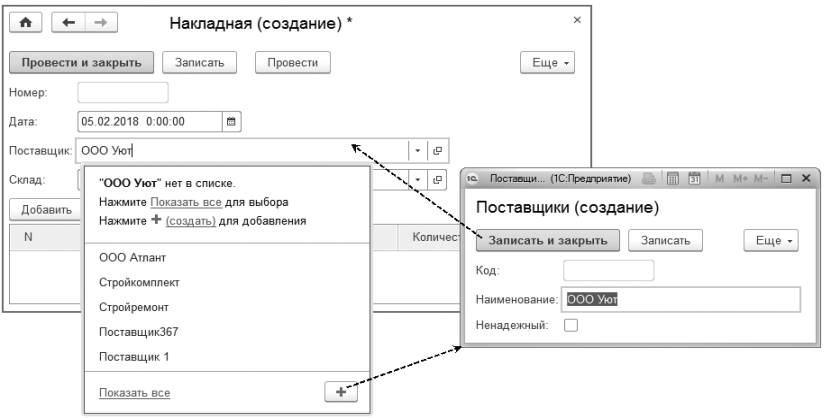


Рис. 3.206. Создание нового элемента справочника при вводе в ссылочное поле

Однако надо учитывать, что пользователь может создать такой элемент данных, который, вообще говоря, не должен оказаться в ссылочном поле. Например, если он не соответствует условиям отбора, которые заданы в свойствах СвязиПараметровВыбора или ПараметрыВыбора этого поля.

Поэтому перед подстановкой нового элемента в ссылочное поле он автоматически проверяется на соответствие параметрам выбора и связям параметров выбора, установленным для этого поля. И если элемент не соответствует условиям отбора, то будет выдано диагностическое сообщение об этом и созданный элемент не будет помещен в поле ввода.

Разработчик может программно вмешаться в процесс проверки, отменить ее или изменить условия проверки и т. д. Для этого в платформе реализован следующий механизм.

При нажатии кнопки создания в выпадающем списке у ссылочного поля возникает событие Создание. В обработчике этого события разработчик может изменить или дополнить условия выбора, заданные в свойствах СвязиПараметровВыбора или ПараметрыВыбора этого поля. Причем он может описать и дополнительные условия выбора, имена которых начинается с префикса «Дополнительно.».

По умолчанию параметр СтандартнаяОбработка события Создание установлен в значение Истина. В этом случае платформа выполнит открытие основной формы объекта в режиме выбора, при этом для осуществления проверки в форму будут переданы параметры:

- ПараметрыВыбора. Эта структура содержит параметры выбора, ограничивающие список элементов, которые могут быть выбраны в поле ввода. Заполняется из параметров выбора и связей параметров выбора. В структуру попадают только те значения, имена которых начинаются с префикса «Отбор.»;
- ДополнительныеПараметры. Эта структура содержит дополнительные параметры, которые разработчик считает нужным передать в форму объекта. Заполняется из параметров выбора и связей параметров выбора. В структуру попадают только те значения, имена которых начинаются с префикса «Дополнительно.».

Владельцем открываемой формы нового объекта будет выступать поле ввода, откуда вызвано создание объекта.

При открытии формы нового объекта значение параметра формы ПараметрыВыбора копируется в одноименное свойство формы. Таким образом, условия отбора выбираемых элементов будут нам доступны через свойство ПараметрыВыбора в течение всей «жизни» формы.

А вот условия, содержащиеся в параметре ДополнительныеПараметры, будут доступны только в обработчике события ПриСозданииНаСервере(), как и все остальные неключевые параметры. Поэтому, если требуется использовать их где-то еще (например, проанализировать их на предмет подстановки созданного значения в поле ввода), нужно сохранять эти условия в специальном реквизите формы.

В момент закрытия формы (после вызова события ПередЗакрытием и перед вызовом события ПриЗакрытии) у формы вызывается событие ВыборЗначения. В обработчике этого события разработчик может выполнить дополнительные проверки или повлиять на результат автоматической проверки с помощью значения свойства ВыборДоступен:

- Если свойство ВыборДоступен установлено в значение Истина, то проверка успешно пройдена и выполняется выбор созданного элемента.

- Если свойство ВыборДоступен установлено в значение Ложь, то формируется сообщение о том, что созданный элемент не соответствует условиям выбора, и выбор созданного элемента не выполняется.

В случае если разработчик хочет самостоятельно выполнить проверку, выдать собственное диагностическое сообщение, поместить значение в поле ввода и т. д., нужно установить параметр СтандартнаяОбработка события ВыборЗначения в значение Ложь.

Примеры можно посмотреть в демонстрационной базе «Поле ввода», форма документа Накладная, формы элементов справочников Поставщики и Товары.

Поясним сказанное на примерах.

Стандартная проверка при выборе значения

В этом примере мы просто программно установим свойство ПараметрыВыбора у ссылочного поля Склад в форме документа Накладная и посмотрим, как будет происходить автоматическая проверка при подстановке в это поле нового склада, созданного с помощью кнопки создания из выпадающего списка.

Перед этим убедимся, что свойство ПараметрыВыбора у реквизита Склад документа Накладная и одноименное свойство у поля Склад, отображающего данные этого реквизита, пусты. Зададим параметры выбора в момент нажатия на кнопку создания в выпадающем списке.

Для этого создадим обработчик события Создание для поля Склад в форме документа Накладная и заполним его следующим образом (листинг 3.161).

Листинг 3.161. Обработчик события «Создание» поля «Склад»

```
&НаКлиенте
Процедура СкладСоздание(Элемент, СтандартнаяОбработка)

    НовыйПараметр = Новый ПараметрВыбора("Отбор.Розничный", Ложь);
    НовыйМассив = Новый Массив();
    НовыйМассив.Добавить(НовыйПараметр);
    НовыеПараметры = Новый ФиксированныйМассив(НовыйМассив);
    Элементы.Склад.ПараметрыВыбора = НовыеПараметры;

КонецПроцедуры
```

Согласно этим параметрам выбора при создании новых элементов из выпадающего списка в поле Склад не должны выбираться розничные склады.

Запустим «1С:Предприятие» и создадим новую накладную. Щелкнем мышью в поле Склад и в открывшемся выпадающем списке нажмем кнопку создания. В появившейся форме нового элемента справочника Склады укажем наименование элемента, установим признак Розничный и нажмем кнопку Записать и закрыть (см. рис. 3.207).

В результате новый элемент справочника будет создан, форма справочника закроется, но созданный элемент не будет помещен в поле ввода. Пользователю будет показано сообщение о том, что созданный элемент не соответствует условиям выбора (рис. 3.207).

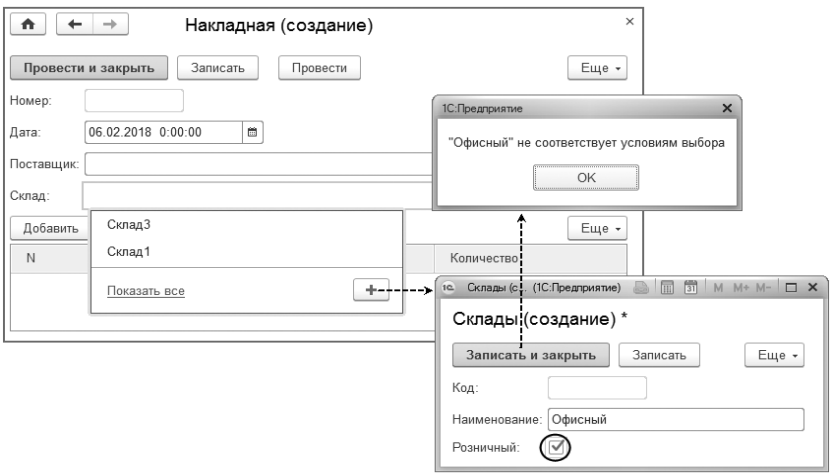


Рис. 3.207. Проверка перед подстановкой нового элемента справочника в поле ввода

Так произошло потому, что перед подстановкой нового элемента в поле ввода вызывается событие ВыборЗначения и платформа выполняет автоматическую проверку созданного значения на соответствие параметрам выбора, содержащимся в свойстве формы ПараметрыВыбора.

Так как мы создали новый склад с установленным признаком Розничный, этот элемент не прошел проверку. В результате

при выходе из обработчика события ВыборЗначения свойство ВыборДоступен платформа установила в значение Ложь.

Обратите внимание, что здесь мы никак не вмешивались в процесс проверки, так как обработчик события, да и сама форма элемента справочника Склады, в конфигурации отсутствуют.

Аналогичного результата можно было бы достигнуть путем интерактивной установки свойства ПараметрыВыбора реквизита Склад или связанного с ним поля формы накладной.

Передача дополнительных параметров выбора в форму нового элемента

В этом примере мы рассмотрим, как при создании нового элемента справочника из выпадающего списка поля ввода передавать дополнительные параметры выбора в форму нового элемента и обрабатывать их в этой форме.

У реквизита Товар табличной части Товары документа Накладная заданы свойства Связи параметров выбора и Параметры выбора (рис. 3.208).

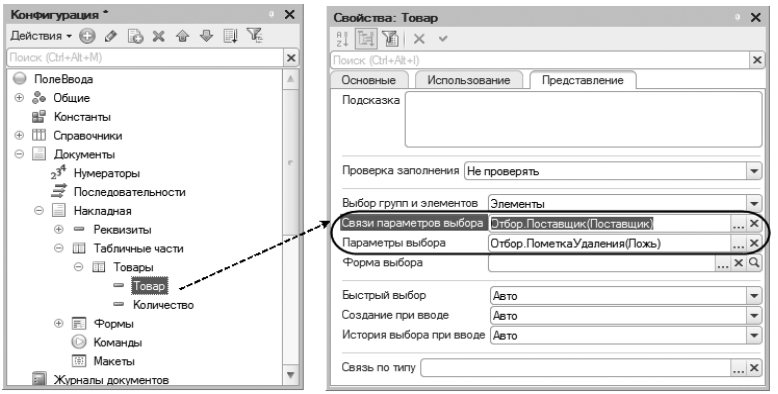


Рис. 3.208. Свойства «Связи параметров выбора» и «Параметры выбора»

В связях параметров выбора указывается, что в поле таблицы формы Товар могут выбираться только товары поставщика, указанного в документе. В параметрах выбора указывается, что эти товары не должны быть помечены на удаление.

Предположим, мы не хотим, чтобы в этом поле выбирались услуги. Зададим это условие в дополнительных параметрах выбора товаров (с префиксом «Дополнительно.») в момент нажатия на кнопку создания в выпадающем списке.

Для этого создадим обработчик события Создание для поля ТоварыТовар в таблице формы Товары и заполним его следующим образом (листинг 3.162).

Листинг 3.162. Обработчик события «Создание» поля «ТоварыТовар»

```
&НаКлиенте
Процедура ТоварыТоварСоздание(Элемент, СтандартнаяОбработка)

    НовыйПараметр = Новый ПараметрВыбора("Дополнительно.Услуга", Ложь);
    НовыйМассив = Новый Массив();
    НовыйМассив.Добавить(НовыйПараметр);
    НовыеПараметры = Новый ФиксированныйМассив(НовыйМассив);
    Элементы.ТоварыТовар.ПараметрыВыбора = НовыеПараметры;

КонецПроцедуры
```

В дополнительных параметрах выбора мы указываем, что у новых товаров, подставляемых в поле таблицы Товар, признак Услуга должен быть выключен. Конечно, мы могли бы задать это условие выбора в качестве отбора в свойстве поля ПараметрыВыбора, причем интерактивно в конфигураторе. Но наш пример демонстрирует работу с дополнительными параметрами выбора, поэтому мы поступаем таким образом.

После этого откроем форму элемента справочника Товары, создадим обработчик события ПриСозданииНаСервере и заполним его следующим образом (листинг 3.163).

Листинг 3.163. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    ВыборУслуг = Истина;
    Параметры.ДополнительныеПараметры.Свойство("Услуга", ВыборУслуг);

    Если ВыборУслуг = Ложь Тогда
        Объект.ВидТовара = Перечисления.ВидыТоваров.Товар;
        Элементы.ВидТовара.Подсказка =
            "Услуги недоступны для выбора при создании из формы накладной";
        Элементы.ВидТовара.ОтображениеПодсказки = ОтображениеПодсказки.Кнопка;
        Элементы.ВидТовара.Доступность = Ложь;
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике в переменной ВыборУслуг мы получаем значение поля Услуга из структуры ДополнительныеПараметры. Если выбор услуг запрещен, то значение реквизита ВидТовара мы устанавливаем в значение Товар перечисления ВидыТоваров. Кроме того, делаем соответствующее поле формы недоступным. И чтобы у пользователей не возникало лишних вопросов, выводим для поля поясняющую подсказку.

Запустим «1С:Предприятие» и создадим новую накладную. Заполним поле Поставщик, добавим строку в таблицу товаров и в открывшемся выпадающем списке под полем Товар нажмем кнопку создания (см. рис. 3.209).

В появившейся форме нового элемента справочника Товары значение поля Вид товара уже задано как Товар (значение перечисления ВидыТоваров), недоступно для изменения и снабжено подсказкой в виде кнопки с пиктограммой вопроса. Укажем наименование элемента, зададим того же поставщика, что и в форме накладной, и нажмем кнопку Записать и закрыть (рис. 3.209).

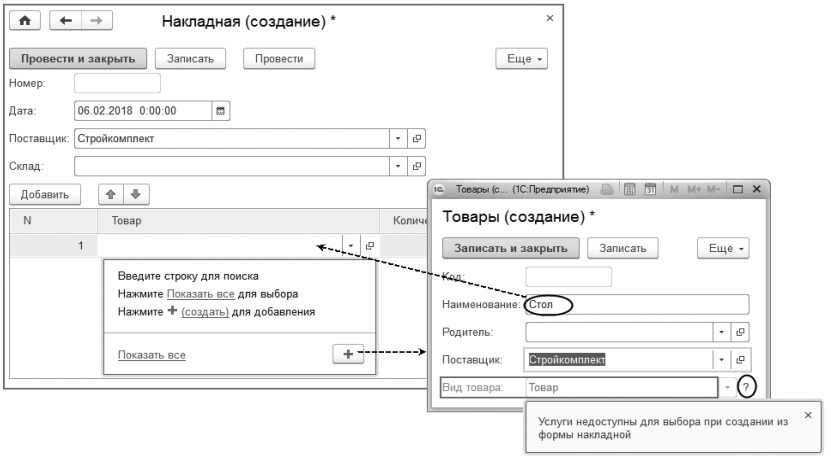


Рис. 3.209. Создание нового элемента справочника при вводе в ссылочное поле

Новый товар будет создан, автоматическая проверка также будет выполнена и успешно пройдена, а ссылка на созданный товар будет подставлена в поле Товар в первую строку таблицы формы накладной.

А вот если в форме создания нового товара указать другого поставщика (отличного от поставщика в накладной), то созданный товар не будет соответствовать связям параметров выбора, установленным в конфигураторе для реквизита накладной Товар (см. рис. 3.208). В результате после автоматической проверки платформа выдаст диагностическое сообщение о том, что новый товар не соответствует условиям выбора, и не подставит его в накладную.

Что касается услуг, то пользователь просто не сможет выбрать этот вид товара при создании из формы накладной. В то время как при создании нового товара из других мест прикладного решения поле Вид товара по-прежнему доступно для изменения и выбор услуг возможен.

Так произошло потому, что дополнительные параметры мы устанавливаем только при нажатии кнопки создания в выпадающем списке у поля таблицы накладной Товар. В остальных случаях в обработчике события ПриСозданииНаСервере() формы нового товара значение параметра ДополнительныеПараметры не определено и выбор услуг разрешен.

Собственная проверка при выборе значения

В этом примере мы выполним собственную проверку перед подстановкой созданного элемента справочника Поставщики в поле ввода накладной Поставщик. Если у нового поставщика установлен признак Ненадежный, то мы отменим стандартную проверку и зададим вопрос, нужно ли подставлять в поле ввода ненадежного поставщика. Если да, то ссылка на ненадежного поставщика, тем не менее, должна попасть в поле ввода.

При нажатии кнопки создания в выпадающем списке у поля накладной Поставщик открывается форма создания нового элемента справочника Поставщики. При этом значения параметров формы справочника ПараметрыВыбора и ДополнительныеПараметры не определены, так как ни программно, ни интерактивно мы их нигде не задавали.

Поэтому, в принципе, любой созданный поставщик может быть подставлен в поле ввода. Но перед этим мы хотим, чтобы пользователь подтвердил свой выбор в случае, если поставщик ненадежный.

Эти действия нужно выполнять в обработчике события ВыборЗначения формы элемента справочника Поставщики. Создадим этот обработчик и заполним его следующим образом (листинг 3.164).

Листинг 3.164. Обработчик события «ВыборЗначения»

```
&НаКлиенте
Процедура ВыборЗначения(СтандартнаяОбработка)

    Если Объект.Ненадежный Тогда
        СтандартнаяОбработка = Ложь;
        ВыборДоступен = Ложь;
        ПоказатьВопрос(Новый ОписаниеОповещения("ВопросЗавершение" ЭтотОбъект),
            "Вы выбрали ненадежного поставщика! Продолжить тем не менее?",
            РежимДиалогаВопрос.ДаНет);
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике события происходит проверка соответствия созданного значения параметрам выбора и связям параметров выбора, установленным для поля ввода, инициировавшего создание нового элемента. В момент обработки события эти условия выбора содержатся в свойстве формы справочника ПараметрыВыбора. И по результатам этой проверки свойство формы ВыборДоступен устанавливается в Истина или в Ложь.

Поскольку ПараметрыВыбора не определены, то проверять нечего и свойство ВыборДоступен при выходе из обработчика всегда будет истинно. Мы вмешиваемся в процесс проверки только в том случае, если у созданного поставщика установлен признак Ненадежный.

В этом случае мы отменяем стандартную обработку события (СтандартнаяОбработка = Ложь), а также устанавливаем результат проверки (свойство ВыборДоступен) в значение Ложь. Если больше ничего не делать, то ненадежный поставщик никогда не попадет в поле ввода накладной.

Но мы хотим, чтобы пользователь сам выбирал, нужно ли подставлять в поле ввода ненадежного поставщика или нет. Поэтому с помощью

немодального метода ПоказатьВопрос() мы задаем ему этот вопрос с вариантами ответа Да или Нет.

Первым параметром в этот метод передается описание оповещения, описывающее экспортную процедуру, расположенную в модуле формы, ВопросЗавершение(), которая будет вызвана, когда пользователь ответит на вопрос (листинг 3.165).

Листинг 3.165. Обработчик события «ВыборЗначения»

```
&НаКлиенте
Процедура ВопросЗавершение(РезультатВопроса, ДополнительныеПараметры) Экспорт

    Если РезультатВопроса = КодВозвратаДиалога.Да Тогда
        ОповеститьОВыборе(Объект.Ссылка);
    КонецЕсли;

КонецПроцедуры
```

В параметре РезультатВопроса процедуры обработки оповещения содержится результат ответа пользователя. Если ответ положительный, значит, пользователь хочет подставить в поле ввода накладной поставщика, несмотря на то что он ненадежный.

В этом случае вызывается метод формы ОповеститьОВыборе() и владельцу формы создания нового элемента справочника – полю ввода накладной Поставщик передается выбранное значение (Объект.Ссылка). В результате у поля ввода будет вызвано событие ОбработкаВыбора и ссылка на поставщика попадет в поле накладной.

Если же пользователь ответил Нет, то больше ничего не произойдет и ссылка на поставщика подставлена не будет, так как стандартная обработка события была отменена.

Запустим «1С:Предприятие» и создадим новую накладную. Щелкнем мышью в поле Поставщик и в открывшемся выпадающем списке нажмем кнопку создания. В появившейся форме нового элемента справочника Поставщики укажем наименование элемента, установим признак Ненадежный и нажмем кнопку Записать и закрыть (см. рис. 3.210).

После этого пользователю будет показано сообщение о том, что он выбрал ненадежного поставщика, с возможностью продолжения

выбора. Если пользователь хочет продолжить, то ссылка на созданного поставщика будет подставлена в поле накладной Поставщик (рис. 3.210).

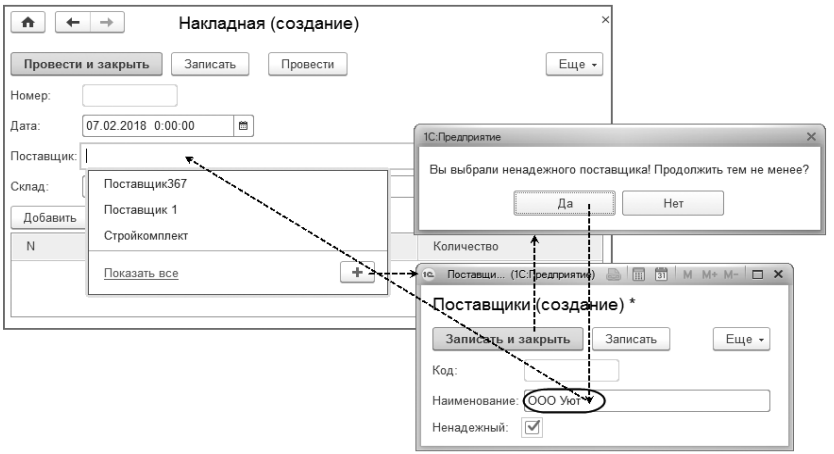


Рис. 3.210. Проверка перед подстановкой нового элемента справочника в поле ввода

Если же пользователь отказался от выбора, то ссылка на поставщика подставлена не будет.

Ну, а если созданный поставщик надежный, все сработает «в штатном режиме», так как в этом случае стандартную обработку события ВыборЗначения мы не отменяли.

Глава 3.20. Программное изменение формы

В «1С:Предприятии» реализован достаточно богатый механизм программного изменения формы, однако он не полностью позволяет реализовать те возможности, которые предоставляются средствами конфигурирования (см. часть 2, «Конструирование форм», стр. 235). Это обусловлено тем, что сама модель прикладного решения ориентирована на работу со статическим, а не с динамическим описанием форм.

Форма не рождается на клиенте. Форма рождается на сервере, она проходит несколько важных стадий предварительной обработки, прежде чем достичь глаз пользователя. Платформа содержит достаточно сложные механизмы многоуровневого кеширования различных частей формы как на сервере, так и на клиенте.

По этой причине возможность программного изменения формы является скорее опциональной, дополнительной. Она рассчитана на отдельные конкретные сценарии работы и не предполагает массового использования в большом количестве форм конфигурации или в часто используемых формах. Основным подходом при разработке прикладных решений должно являться визуальное конструирование форм в конфигураторе. А программное изменение форм рекомендуется использовать лишь в отдельных специфических сценариях работы.

Такими сценариями могут быть, например, отображение в форме имеющихся типовых операций или характеристик объектов. То есть той информации, которая содержится в базе данных и структура которой неизвестна на этапе конфигурирования. Ее можно узнать только уже в процессе функционирования прикладного решения, в режиме 1С:Предприятие. Поэтому для ее отображения в форме и требуется ее программное изменение.

Общие подходы

Встроенный язык позволяет добавлять, изменять и удалять реквизиты, команды и элементы формы. Особенность заключается в том, что программно удалить можно только то, что программно же и добавлено. Нельзя программно удалить элементы, реквизиты или команды, созданные в конфигураторе.

При программном изменении формы нужно управлять всей «троицей»: *реквизиты*, *команды* и *элементы*. Например, чтобы разместить какие-то данные в форме, нужно создать реквизит, создать элемент, связать элемент с реквизитом. Для команды – создать команду, связать ее с имеющимся в модуле формы обработчиком, создать элемент, связать элемент с командой.

Добавление и удаление реквизитов, команд и элементов формы возможно только на сервере. Поэтому алгоритмы модификации формы нужно размещать в серверных процедурах формы – в обработчиках событий формы на сервере или в контекстных серверных процедурах модуля формы.

Программное изменение формы возможно только «изнутри» формы, т. е. при нахождении в ее модуле. Нет возможности получить какую-либо форму, программно изменить ее и затем открыть эту форму. Потому что изменение формы возможно только на сервере, а открытие формы – только на клиенте. А встроенный язык не содержит методов, позволяющих разработчику принудительно передать форму с сервера на клиент.

Механика добавления/удаления реквизитов формы принципиально отличается от механики работы с ее элементами и командами.

Элементы и команды можно добавлять/удалять поодиночке, просто обращаясь к коллекциям этих элементов, используя методы `Добавить()` и `Удалить()`. Это относительно «безболезненные» операции для формы.

Изменение состава реквизитов, напротив, является сложной и затратной операцией. Поэтому здесь используется следующий подход. Сначала разработчик создает два массива программных объектов, которые описывают реквизиты формы. Один массив – это

те реквизиты, которые должны быть добавлены, а другой массив – это те реквизиты, которые нужно удалить. После этого «за один подход» выполняется модификация формы с помощью ее метода `ИзменитьРеквизиты()`, в который передаются оба этих массива. Сначала выполняется удаление реквизитов, затем – добавление.

С добавлением реквизитов связана еще одна важная особенность. Можно добавить реквизит и установить его свойства. Но из встроенного языка нельзя назначить реквизит основным. Поэтому, например, полностью программно невозможно создать «настоящую» форму списка или объекта.

Теперь в качестве примеров рассмотрим четыре случая программного изменения формы:

- добавление поля,
- добавление динамического списка,
- добавление колонки в таблицу,
- добавление команды.

Добавление поля

Разберем пример с добавлением поля так, как мы делали бы его в конфигураторе, –средствами визуального конструирования.

Сначала мы добавим реквизит формы типа `Строка` (рис. 3.211).

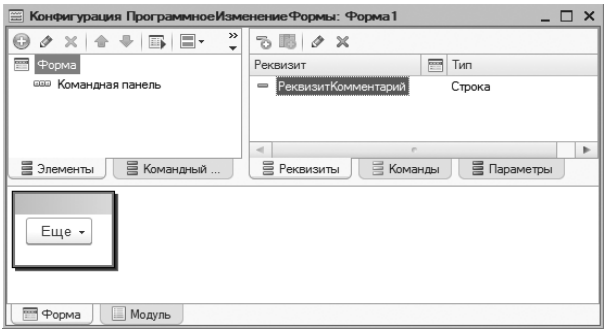


Рис. 3.211. Реквизит формы «РеквизитКомментарий»

После этого добавим элемент формы Поле, который в дальнейшем будет отображать значение реквизита (рис. 3.212).

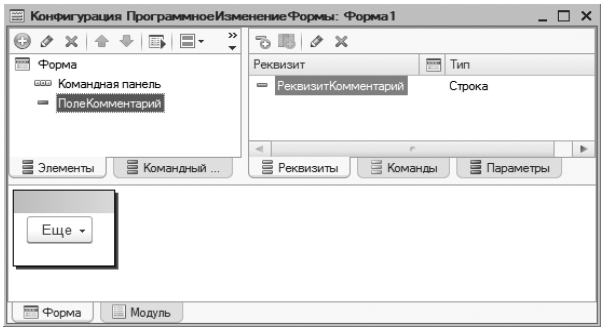


Рис. 3.212. Поле «ПолеКомментарий»

Затем свяжем между собой элемент формы и реквизит формы (рис. 3.213).

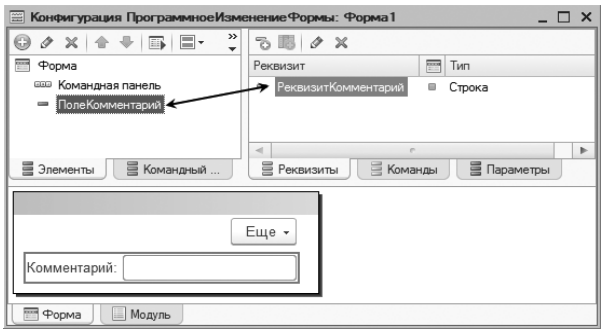


Рис. 3.213. Связь поля и реквизита

И в заключение для элемента формы назначим обработчик события При изменении, который заранее заготовлен в модуле нашей формы (рис. 3.214).

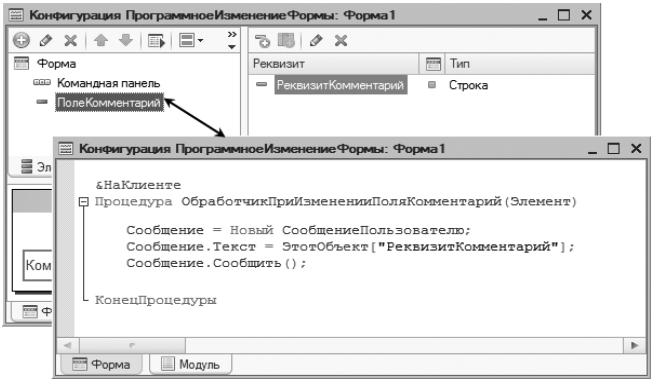


Рис. 3.214. Назначение обработчика события «При изменении»

Теперь все те же самые действия выполним программно, из встроенного языка.

Для изменения формы сразу же выполним контекстный вызов сервера, в котором и будем модифицировать форму (листинг 3.166).

Пример можно посмотреть в демонстрационной базе «Программное изменение формы», общая форма Форма, команда Добавить поле. Это локальная команда этой формы.

Листинг 3.166. Контекстный вызов сервера для модификации формы

```
&НаКлиенте
Процедура ДобавитьПоле(Команда)

        ДобавитьПолеНаСервере();

КонецПроцедуры

&НаСервере
Процедура ДобавитьПолеНаСервере()
...
```

В серверной процедуре сначала добавим реквизит формы РеквизитКомментарий (листинг 3.167).

Листинг 3.167. Добавление реквизита формы

```
// Добавить реквизит.
ТипыРеквизита = Новый Массив;
ТипыРеквизита.Добавить(Тип("Строка"));

ОписаниеТиповДляРеквизита = Новый ОписаниеТипов(ТипыРеквизита);

НовыйРеквизит = Новый РеквизитФормы("РеквизитКомментарий",  // имя
        ОписаниеТиповДляРеквизита,              // тип
        ,                                       // путь
        "Комментарий",                          // заголовок
        Истина);                                // сохраняемые данные

ДобавляемыеРеквизиты = Новый Массив;
ДобавляемыеРеквизиты.Добавить(НовыйРеквизит);

ИзменитьРеквизиты(ДобавляемыеРеквизиты);
```

В этом фрагменте мы сначала создаем объект, описывающий тип добавляемого реквизита. Поскольку в общем случае реквизит может иметь составной тип, т. е. включать в себя сразу несколько типов, то для описания его типа используется объект `ОписаниеТипов`. Этот объект создается на основе массива, содержащего нужные типы.

Наш реквизит будет иметь тип `Строка`, поэтому мы создаем массив `ТипыРеквизита`, содержащий единственный элемент. Затем на основании этого массива создаем описание типов – `ОписаниеТиповДляРеквизита`. Это описание типов мы используем при конструировании реквизита формы – `НовыйРеквизитФормы(…)`.

В параметрах конструктора мы указываем:

- Имя реквизита – `РеквизитКомментарий`.
- Тип этого реквизита – созданное нами описание типов `ОписаниеТиповДляРеквизита`.
- Путь к реквизиту не указываем, поскольку добавляем реквизит первого уровня.
- Заголовок реквизита – `Комментарий`, этот заголовок будет использоваться платформой для отображения в форме.
- Последним параметром указываем, что реквизит содержит сохраняемые данные, т. е. при изменении этого реквизита будет автоматически устанавливаться модифицированность формы.

В данном случае этот параметр мы используем исключительно в демонстрационных целях, чтобы показать, что такая возможность имеется. Благодаря этому в дальнейшем перед закрытием формы мы сможем анализировать ее модифицированность и предотвращать закрытие формы без сохранения измененных данных.

После того как реквизит создан, мы добавляем его в массив и этот массив передаем в метод формы ИзменитьРеквизиты() первым параметром. То есть как те реквизиты, которые нужно добавить в форму.

Теперь мы имеем состояние, аналогичное тому, которое было показано на рисунке 3.211.

На втором этапе, как мы говорили выше, нужно создать элемент формы. Для этого мы используем следующий код (листинг 3.168).

Листинг 3.168. Добавление элемента формы

```
// Добавить элемент формы и связать его с реквизитом.
НовыйЭлемент = Элементы.Добавить("ПолеКомментарий", Тип("ПолеФормы"));
```

В результате мы имеем состояние, которое было показано на рисунке 3.212.

Теперь мы связываем элемент формы и реквизит. Для этого в свойстве элемента ПутьКДанным мы указываем имя нашего реквизита (листинг 3.169).

Листинг 3.169. Связь элемента с реквизитом

```
НовыйЭлемент.ПутьКДанным = "РеквизитКомментарий";
```

Теперь мы имеем состояние, которое было показано на рисунке 3.213.

После того как элемент добавлен и связан с реквизитом, установим некоторые его свойства. Во-первых, укажем, что это не просто поле, а поле ввода, а во-вторых – отобразим в нем кнопку очистки (листинг 3.170).

Листинг 3.170. Установка свойств поля

```
НовыйЭлемент.Вид = ВидПоляФормы.ПолеВвода;
НовыйЭлемент.КнопкаОчистки = Истина;
```

Теперь нам осталось выполнить последний шаг – назначить обработчик события для только что добавленного элемента формы (листинг 3.171).

Листинг 3.171. Назначение обработчика события

```
// Установить обработчик события.
НовыйЭлемент.УстановитьДействие("ПриИзменении", "ОбработчикПриИзмененииПоляКомментарий");
```

Здесь мы указываем имя события и имя процедуры, которая будет обрабатывать это событие. Эта процедура должна быть создана в модуле формы заранее, на этапе конфигурирования (листинг 3.172).

Листинг 3.172. Обработчик события «При изменении»

```
&НаКлиенте
Процедура ОбработчикПриИзмененииПоляКомментарий(Элемент)

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = ЭтотОбъект["РеквизитКомментарий"];
    Сообщение.Сообщить();

КонецПроцедуры
```

После этого можно запустить систему в режиме 1С:Предприятие и посмотреть, как работает наша команда Добавить поле.

В результате ее выполнения внизу формы появится поле ввода Комментарий. При изменении значения этого поля будет обрабатываться событие, и новое значение поля будет выводиться в окно сообщений (рис. 3.215).

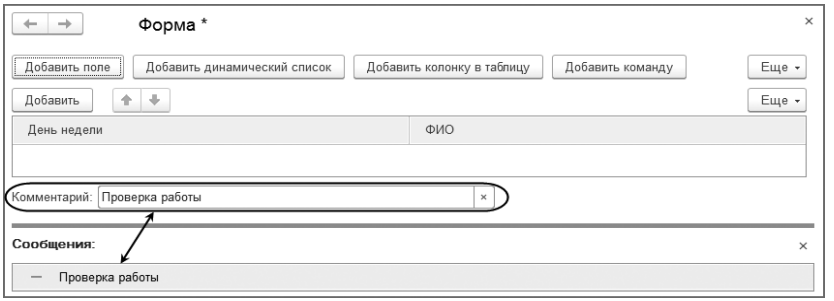


Рис. 3.215. Поле ввода в форме

Добавление динамического списка

Пример с добавлением динамического списка, если бы мы выполняли его средствами визуального конструирования, выглядел бы следующим образом.

Сначала мы добавим реквизит формы типа ДинамическийСписок (рис. 3.216).

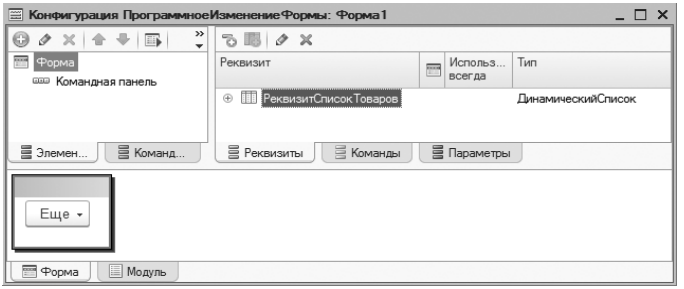


Рис. 3.216. Добавление реквизита формы

После этого для динамического списка установим флажок Произвольный запрос и зададим текст запроса и основную таблицу (рис. 3.217).

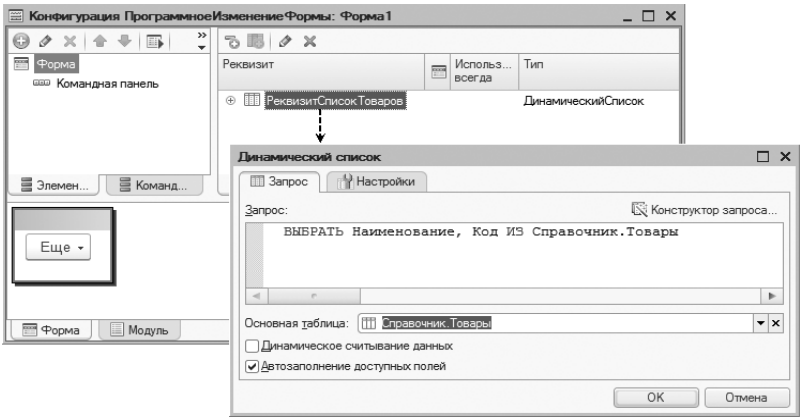


Рис. 3.217. Установка текста запроса и основной таблицы

Затем добавим элемент формы Таблица (рис. 3.218).

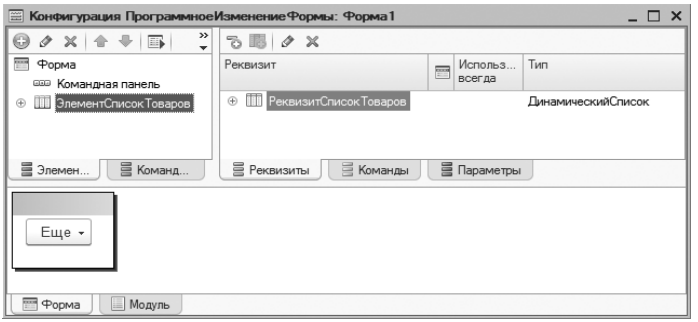


Рис. 3.218. Добавление таблицы формы

После этого свяжем между собой элемент формы и реквизит формы (рис. 3.219).

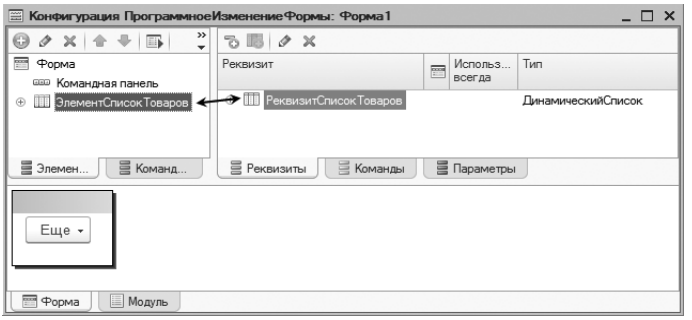


Рис. 3.219. Связь таблицы и реквизита

Теперь создадим колонку таблицы Наименование и свяжем ее с колонкой динамического списка Наименование (рис. 3.220).

Аналогичным образом создадим еще одну колонку таблицы Код и свяжем ее с колонкой динамического списка Код (рис. 3.221).

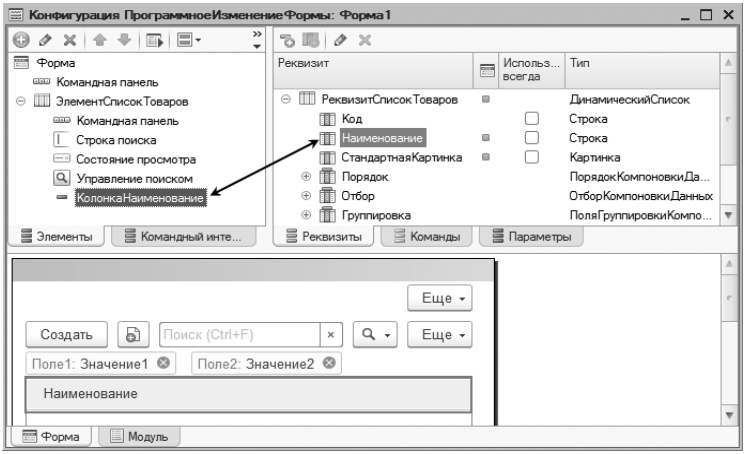


Рис. 3.220. Добавление колонки «КолонкаНаименование» и ее связь с полем реквизита

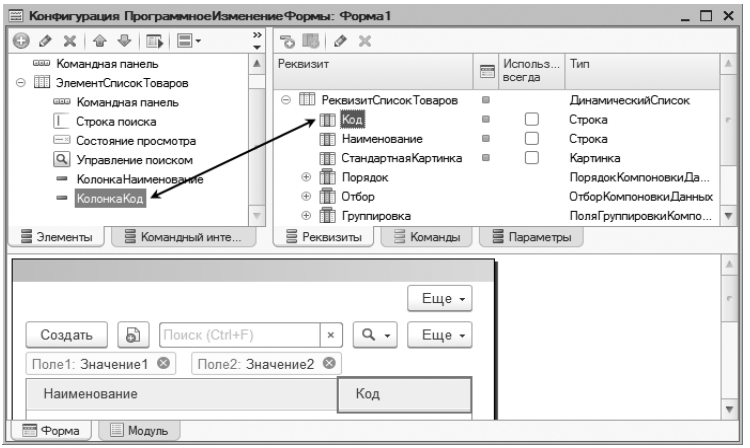


Рис. 3.221. Добавление колонки «КолонкаКод» и ее связь с полем реквизита

Теперь эти же самые действия выполним программно, из встроенного языка.

Как и в предыдущем примере, модификацию формы будем выполнять в серверной контекстной процедуре.

Пример можно посмотреть в демонстрационной базе «Программное изменение формы», общая форма Форма, команда Добавить динамический список. Это локальная команда этой формы.

Сначала добавим реквизит формы `РеквизитСписокТоваров` (листинг 3.173).

Листинг 3.173. Добавление реквизита формы

```
// Добавить реквизит "Динамический список".
ТипыРеквизита = Новый Массив;
ТипыРеквизита.Добавить(Тип("ДинамическийСписок"));

ОписаниеТиповДляРеквизита = Новый ОписаниеТипов(ТипыРеквизита);

НовыйРеквизит = Новый РеквизитФормы("РеквизитСписокТоваров",// имя
        ОписаниеТиповДляРеквизита,      // тип
        ,                               // путь
        "СписокТоваров",                // заголовок
        Истина);                        // сохраняемые данные

ДобавляемыеРеквизиты = Новый Массив;
ДобавляемыеРеквизиты.Добавить(НовыйРеквизит);

ИзменитьРеквизиты(ДобавляемыеРеквизиты);
```

Здесь все аналогично предыдущему примеру, за исключением типа добавляемого реквизита – `ТипыРеквизита.Добавить(Тип("ДинамическийСписок"))`.

В результате мы имеем состояние, показанное на рисунке 3.216.

Теперь, после того как реквизит добавлен, можно задать текст запроса и основную таблицу динамического списка (листинг 3.174).

Листинг 3.174. Установка текста запроса и основной таблицы

```
// Задать текст запроса и другие свойства динамического списка.
РеквизитСписок = ЭтотОбъект["РеквизитСписокТоваров"];
РеквизитСписок.ТекстЗапроса = "ВЫБРАТЬ Наименование, Код ИЗ Справочник.Товары";
РеквизитСписок.ОсновнаяТаблица = "Справочник.Товары";
```

В результате мы имеем состояние, показанное на рисунке 3.217.

Теперь добавим элемент формы и свяжем его с данными. Снова все, как и в предыдущем примере. Отличается лишь тип добавляемого элемента (листинг 3.175).

Листинг 3.175. Добавление элемента формы

```
// Добавить элемент формы и связать его с реквизитом.
НовыйЭлемент = Элементы.Добавить("ЭлементСписокТоваров", Тип("ТаблицаФормы"));
НовыйЭлемент.ПутьКДанным = "РеквизитСписокТоваров";
```

В результате мы имеем состояние, показанное на рисунке 3.219.

Особенность работы с таблицей заключается в том, что при визуальном конструировании платформа предлагает автоматически создать колонки таблицы, если вы мышью перетаскиваете реквизит в дерево элементов формы.

В нашем же случае необходимо это сделать самостоятельно. И не только создать колонки, но и связать их с данными (листинг 3.176).

Листинг 3.176. Создание колонок и их связь с данными

```
// Создать колонки и связать их с данными.
НоваяКолонкаТаблицы = Элементы.Добавить("КолонкаНаименование", Тип("ПолеФормы"),
        НовыйЭлемент);
НоваяКолонкаТаблицы.ПутьКДанным = "РеквизитСписокТоваров.Наименование";

НоваяКолонкаТаблицы = Элементы.Добавить("КолонкаКод", Тип("ПолеФормы"), НовыйЭлемент);
НоваяКолонкаТаблицы.ПутьКДанным = "РеквизитСписокТоваров.Код";
```

Тут особенность заключается в том, что, во-первых, при добавлении колонки (поля формы) мы указываем его родителя третьим параметром метода `Добавить()`. `НовыйЭлемент` – это как раз та самая таблица, которую мы только что добавили в форму.

Во-вторых, указывая путь к данным, мы указываем его полностью, от корня реквизитов. Сначала имя реквизита, являющегося динамическим списком, а потом через точку – имя одного из полей запроса, содержащегося в этом динамическом списке – `РеквизитСписокТоваров.Наименование`.

В результате мы имеем состояние, показанное на рисунке 3.221.

После этого можно запустить систему в режиме 1С:Предприятие и посмотреть, как работает наша команда Добавить динамический список.

В результате ее выполнения внизу формы появится полноценный список товаров, позволяющий добавлять, изменять и удалять товары (рис. 3.222).

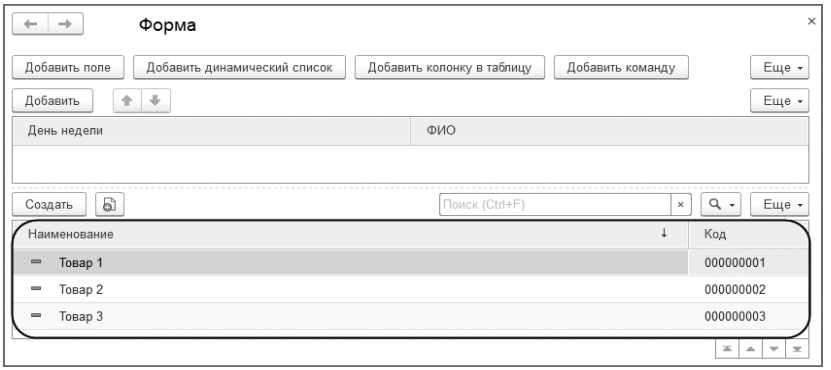


Рис. 3.222. Динамический список в форме

Добавление колонки в таблицу

Пример с добавлением колонки в существующую таблицу, если бы мы выполняли его средствами визуального конструирования, выглядел бы следующим образом.

Сначала мы добавим подчиненный реквизит формы типа `Строка` (рис. 3.223).

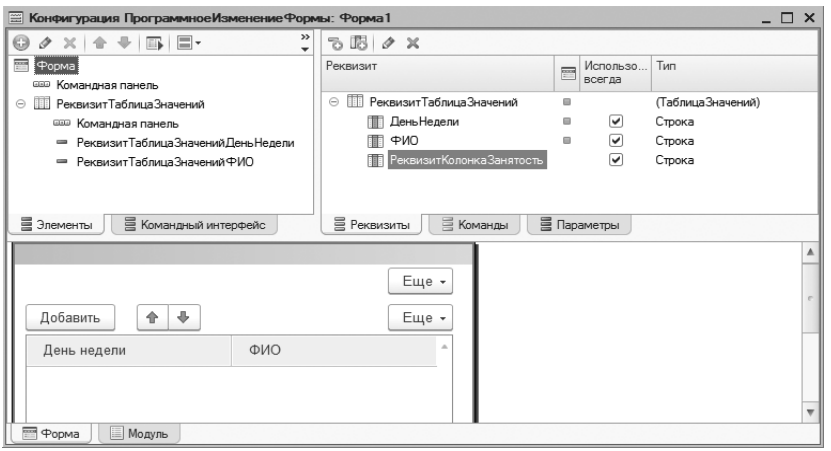


Рис. 3.223. Добавление реквизита формы

После этого добавим элемент формы Поле как подчиненный элемент существующей в форме таблицы (рис. 3.224).

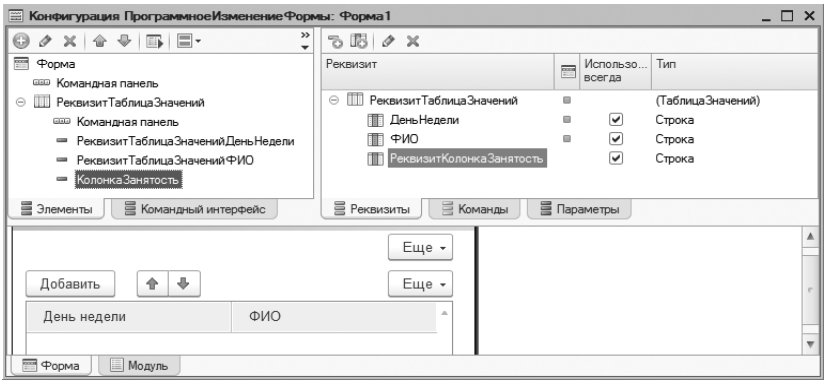


Рис. 3.224. Добавление элемента формы

Затем свяжем между собой элемент формы и реквизит формы (рис. 3.225).

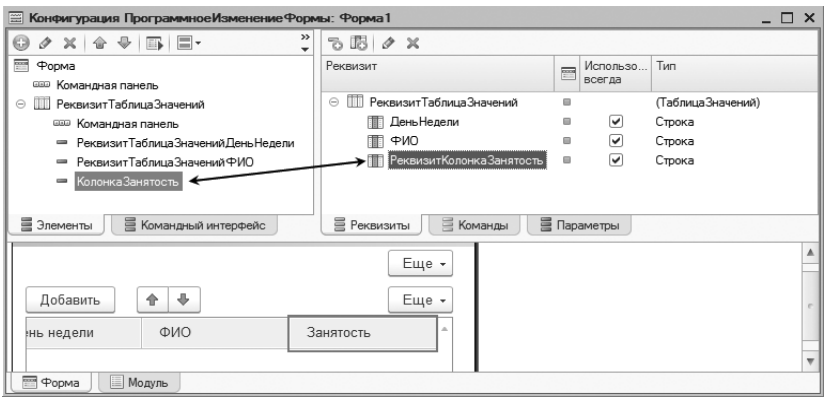


Рис. 3.225. Связь элемента и реквизита

В заключение для элемента формы назначим обработчик события При изменении, который заранее заготовлен в модуле нашей формы (рис. 3.226).

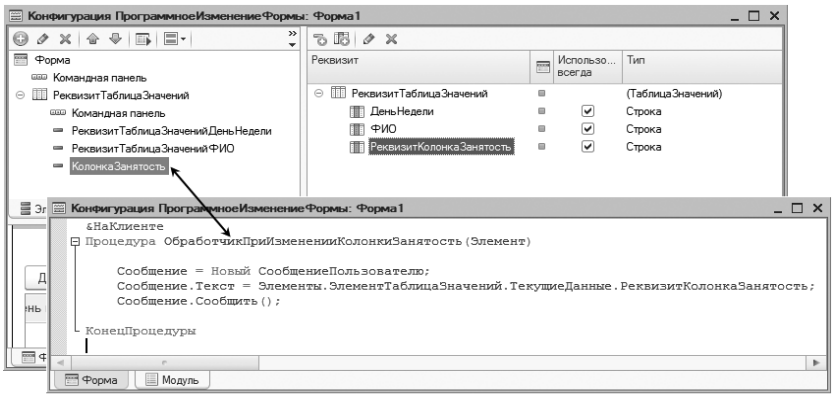


Рис. 3.226. Назначение обработчика события

Теперь все те же самые действия выполним программно, из встроенного языка.

Как и раньше, модификацию формы будем выполнять в серверной контекстной процедуре. Сначала добавим реквизит формы `РеквизитКолонкаЗанятость` (листинг 3.177).

Пример можно посмотреть в демонстрационной базе «Программное изменение формы», общая форма `Форма`, команда `Добавить колонку в таблицу`. Это локальная команда этой формы.

Листинг 3.177. Добавление реквизита формы

```
// Добавить реквизит.
ТипыРеквизита = Новый Массив;
ТипыРеквизита.Добавить(Тип("Строка"));

ОписаниеТиповДляРеквизита = Новый ОписаниеТипов(ТипыРеквизита);

НовыйРеквизит = Новый РеквизитФормы("РеквизитКолонкаЗанятость",      // имя
        ОписаниеТиповДляРеквизита,      // тип
        "РеквизитТаблицаЗначений",      // путь
        "Занятость",                    // заголовок
        Истина);                                // сохраняемые данные

ДобавляемыеРеквизиты = Новый Массив;
ДобавляемыеРеквизиты.Добавить(НовыйРеквизит);

ИзменитьРеквизиты(ДобавляемыеРеквизиты);
```

Здесь особенность в том, что при добавлении реквизита мы указываем параметр Путь – РеквизитТаблицаЗначений. Поскольку добавляемый реквизит подчинен существующему в форме реквизиту, в этом параметре нужно указать полный путь от корня реквизитов до добавляемого нами реквизита, не включая имя самого добавляемого реквизита.

В результате мы имеем состояние, показанное на рисунке 3.223.

После этого добавим элемент формы (листинг 3.178).

Листинг 3.178. Добавление элемента формы

```
// Добавить элемент формы и связать его с реквизитом.
НовыйЭлемент = Элементы.Добавить("КолонкаЗанятость", Тип("ПолеФормы"),
        Элементы.ЭлементТаблицаЗначений);
```

Здесь та же особенность. При добавлении элемента мы указываем его родителя – элемент Таблица, существующий в форме Элементы.ЭлементТаблицаЗначений.

В результате мы имеем состояние, показанное на рисунке 3.224.

Теперь свяжем элемент формы с реквизитом формы и установим некоторые свойства элемента формы (листинг 3.179).

Листинг 3.179. Связь элемента с реквизитом и установка свойств элемента

```
НовыйЭлемент.ПутьКДанным = "РеквизитТаблицаЗначений.РеквизитКолонкаЗанятость";
НовыйЭлемент.Вид = ВидПоляФормы.ПолеВвода;
НовыйЭлемент.КнопкаОчистки = Истина;
```

В результате мы имеем состояние, показанное на рисунке 3.225.

В заключение назначим обработчик события для только что добавленного элемента формы (листинг 3.180).

Листинг 3.180. Назначение обработчика события

```
// Установить обработчик события.
НовыйЭлемент.УстановитьДействие("ПриИзменении", "ОбработчикПриИзмененииКолонкиЗанятость");
```

В результате мы имеем состояние, показанное на рис. 3.226.

После этого можно запустить систему в режиме 1С:Предприятие и посмотреть, как работает наша команда Добавить колонку в таблицу.

В результате ее выполнения в существующей в форме таблице появится колонка Занятость. А когда мы изменим данные в этой колонке, новое значение поля будет выведено в окно сообщений – отработает назначенный нами обработчик При изменении (рис. 3.227).

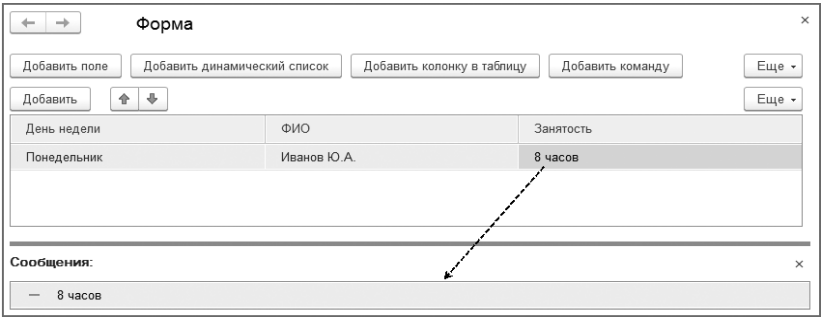


Рис. 3.227. Дополнительная колонка в таблице значений

Добавление команды

Последний пример, который мы рассмотрим, – это добавление команды. Если бы мы выполняли его в конфигураторе, это выглядело бы следующим образом.

Сначала мы добавим команду формы НоваяКоманда (рис. 3.228).

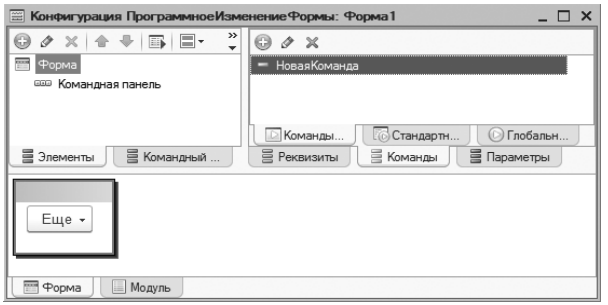


Рис. 3.228. Добавление команды

Затем мы свяжем эту команду с обработчиком, который будет вызываться при исполнении этой команды (рис. 3.229).

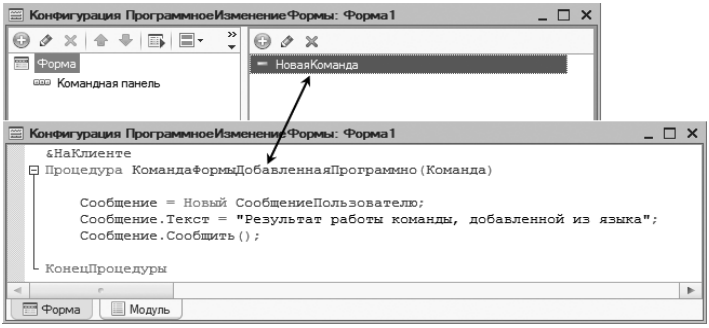


Рис. 3.229. Назначение обработчика команды

После этого добавим в командную панель формы кнопку (рис. 3.230).

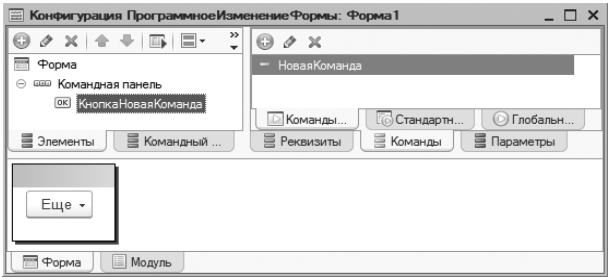


Рис. 3.230. Добавление кнопки в командную панель

И в заключение свяжем кнопку командной панели с той командой, которую мы добавили (рис. 3.231).

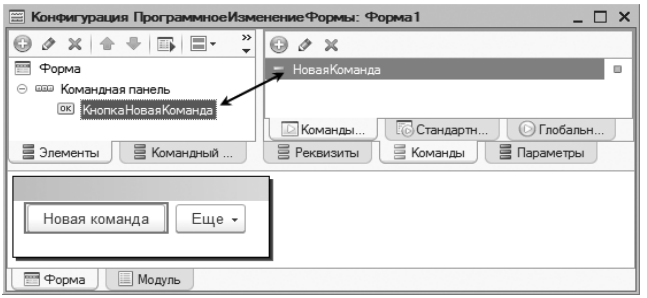


Рис. 3.231. Связь кнопки с командой

Теперь все те же самые действия выполним программно, из встроенного языка.

Как и раньше, модификацию формы будем выполнять в серверной контекстной процедуре. Сначала добавим команду НоваяКоманда (листинг 3.181).

Пример можно посмотреть в демонстрационной базе «Программное изменение формы», общая форма Форма, команда Добавить команду. Это локальная команда этой формы.

Листинг 3.181. Добавление команды

```
// Добавить команду.
НоваяКоманда = Команды.Добавить("НоваяКоманда");
```

В результате мы имеем состояние, показанное на рисунке 3.228.

Назначим этой команде обработчик – процедуру, которая была создана в модуле формы заранее, на этапе конфигурирования (листинг 3.182).

Листинг 3.182. Назначение обработчика команды

```
НоваяКоманда.Действие = "КомандаФормыДобавленнаяПрограммно";
```

В результате мы имеем состояние, показанное на рисунке 3.229.

Добавим кнопку в командную панель формы (листинг 3.183).

Листинг 3.183. Добавление кнопки в командную панель

```
// Добавить кнопку и связать ее с командой.
НовыйЭлемент = Элементы.Добавить("КнопкаНоваяКоманда", Тип("КнопкаФормы"),
        Элементы.ФормаКомандняяПанель);
```

Здесь мы указываем родителя для добавляемой кнопки – существующую в форме командную панель Элементы.ФормаКомандняяПанель.

В результате мы имеем состояние, показанное на рисунке 3.230.

Свяжем кнопку с командой и укажем, что кнопка будет являться кнопкой по умолчанию (она будет выделена среди других кнопок командной панели) – листинг 3.184.

Листинг 3.184. Связь кнопки с командой

```
НовыйЭлемент.ИмяКоманды = "НоваяКоманда";
НовыйЭлемент.КнопкаПоУмолчанию = Истина;
```

В результате мы имеем состояние, показанное на рисунке 3.231.

После этого можно запустить систему в режиме 1С:Предприятие и посмотреть, как работает наша команда Добавить команду.

В результате ее выполнения в командной панели формы появится кнопка Новая команда. А при нажатии этой кнопки в окно сообщений будет выведен отладочный текст (рис. 3.232).

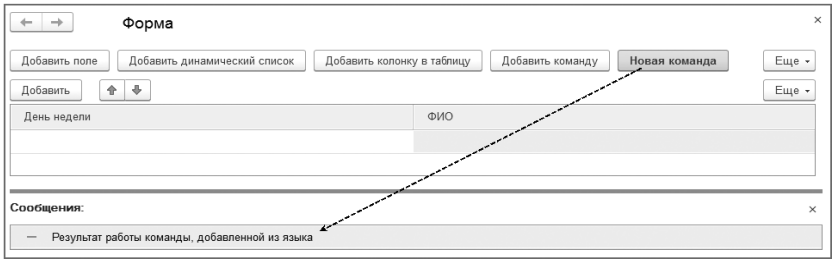


Рис. 3.232. Кнопка в форме

Глава 3.21. Программная настройка интерфейса

Как уже говорилось раньше, состав и расположение панелей интерфейса в основном окне приложения, а также состав и расположение форм на начальной странице могут быть настроены интерактивно. Сначала разработчиком в конфигураторе для всех пользователей прикладного решения, а затем отдельным пользователем для себя в режиме 1С:Предприятие.

Кроме того, эти же настройки можно выполнить из встроенного языка при старте прикладного решения. Например, если в зависимости от роли пользователя нужно программным образом настраивать его рабочее пространство.

Пример можно посмотреть в демонстрационной базе «Открытие форм», модуль управляемого приложения и общий модуль РаботаСИнтерфейсом.

Настройка состава панелей интерфейса

Основной объект, с помощью которого выполняется настройка панелей интерфейса, – НастройкиИнтерфейсаКлиентскогоПриложения. Он позволяет работать как с настройками, заданными в конфигураторе разработчиком, так и с настройками, которые пользователь установил для себя в режиме 1С:Предприятие.

Процедуру настройки интерфейса нужно вызывать в модуле управляемого приложения, в обработчике события ПриНачалеРаботыСистемы (листинг 3.185).

Листинг 3.185. Обработчик события «ПриНачалеРаботыСистемы»

```
Процедура ПриНачалеРаботыСистемы()

    РаботаСИнтерфейсом.НастройкаИнтерфейса();
    ОбновитьИнтерфейс();

КонецПроцедуры
```

После выполнения настройки необходимо выполнить перестроение интерфейса клиентского приложения с помощью метода глобального контекста ОбновитьИнтерфейс().

Саму процедуру настройки интерфейса поместим в общий неглобальный модуль РаботаСИнтерфейсом с установленными свойствами Сервер и Вызов сервера (листинг 3.186).

Листинг 3.186. Процедура «НастройкаИнтерфейса»

```
Процедура НастройкаИнтерфейса() Экспорт

    ПользовательИБ = ПользователиИнформационнойБазы.ТекущийПользователь();
    Для Каждого РольПользователя Из ПользовательИБ.Роли Цикл
        Если РольПользователя.Имя = "Администратор" Тогда
            РасширенныйИнтерфейс();
        КонецЕсли;
        Если РольПользователя.Имя = "Менеджер" Тогда
            МинимальныйИнтерфейс();
        КонецЕсли;
    КонецЦикла;

КонецПроцедуры
```

В этой процедуре для пользователя с ролью Менеджер мы вызываем процедуру для формирования минималистичного интерфейса, а для пользователя с ролью Администратор вызываем процедуру формирования расширенного интерфейса. Для простоты будем считать, что каждому пользователю присвоена только одна роль.

Для менеджера мы будем отображать в основном окне приложения только панель инструментов и панель открытых, причем независимо от его личных настроек и настроек, сделанных в конфигураторе.

Поместим в общем модуле процедуру `МинимальныйИнтерфейс()` и заполним ее следующим образом (листинг 3.187).

Листинг 3.187. Процедура «МинимальныйИнтерфейс»

```
Процедура МинимальныйИнтерфейс() Экспорт

    НастройкиИнтерфейса = Новый НастройкиИнтерфейсаКлиентскогоПриложения;
    НастройкиСостава = НастройкиИнтерфейса.ПолучитьСостав();

    // Очистить настройки состава.
    НастройкиСостава.Верх.Очистить();
    НастройкиСостава.Лево.Очистить();
    НастройкиСостава.Низ.Очистить();
    НастройкиСостава.Право.Очистить();

    ПанельИнструментов = Новый
        ЭлементНастройкиСоставаИнтерфейсаКлиентскогоПриложения("ПанельИнструментов");
```

```
    ПанельОткрытых = Новый
        ЭлементНастройкиСоставаИнтерфейсаКлиентскогоПриложения("ПанельОткрытых");

    НастройкиСостава.Верх.Добавить(ПанельИнструментов);
    НастройкиСостава.Низ.Добавить(ПанельОткрытых);

    НастройкиИнтерфейса.УстановитьСостав(НастройкиСостава);
    ХранилищеСистемныхНастроек.Сохранить(
        "Общее/НастройкиИнтерфейсаКлиентскогоПриложения",, НастройкиИнтерфейса);

КонецПроцедуры
```

В этой процедуре мы конструктором создаем пустой объект `НастройкиИнтерфейсаКлиентскогоПриложения`. Затем с помощью метода этого объекта `ПолучитьСостав()` мы получаем настройки состава интерфейса, установленные в конфигураторе. После этого очищаем список панелей интерфейса и групп панелей, размещенных в верхней, левой, правой и нижней частях основного окна клиентского приложения.

Это необходимо сделать, так как, даже если разработчик ничего не настраивал в конфигураторе, стандартно в интерфейсе клиентского приложения уже отображаются панель разделов, панель функций текущего раздела и панель инструментов (рис. 3.233).

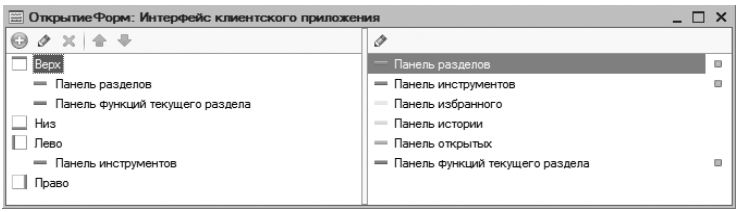


Рис. 3.233. Стандартный состав панелей интерфейса в конфигураторе

Затем создаем конструктором новые элементы настройки состава интерфейса на основании стандартных имен панелей – `ПанельИнструментов` и `ПанельОткрытых`. И добавляем их соответственно в верхнюю и нижнюю часть основного окна приложения.

После этого с помощью метода `УстановитьСостав()` мы загружаем измененные настройки состава интерфейса в пустой объект `НастройкиИнтерфейсаКлиентскогоПриложения`.

И в заключение сохраняем эти настройки для текущего пользователя в хранилище системных настроек с ключом `Общее/НастройкиИнтерфейсаКлиентскогоПриложения`. Это необходимо сделать, так как пользовательские настройки применяются поверх всех остальных и затирают их.

Запустим «1С:Предприятие» от имени пользователя с ролью Менеджер. Независимо от настроек, сделанных в конфигураторе, и от личных настроек менеджера, сделанных в прошлом сеансе работы, при старте прикладного решения для него всегда будет отображаться минималистичный интерфейс, состоящий из рабочей области, панели инструментов (сверху) и панели открытых в нижней части основного окна приложения (рис. 3.234).

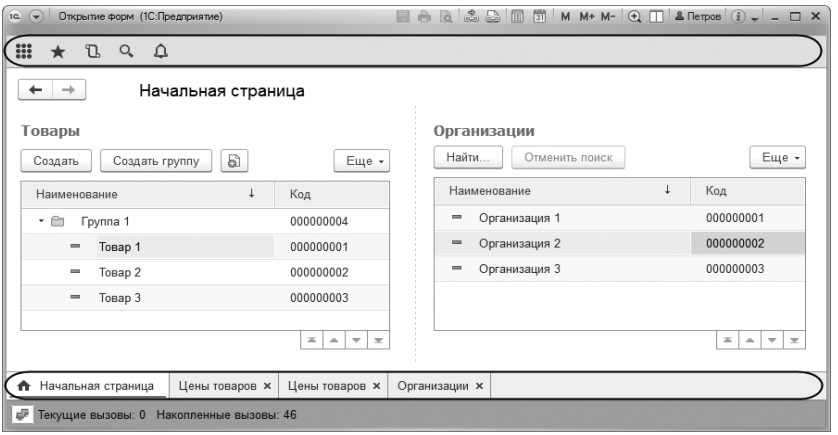


Рис. 3.234. Минимальный состав панелей интерфейса для менеджера

Теперь рассмотрим формирование состава панелей интерфейса для пользователя с ролью Администратор на основе настроек, сделанных им в режиме 1С:Предприятие.

Предположим, администратор в пользовательском режиме изменил стандартное расположение панелей интерфейса. Он скрыл панель разделов и установил отображение панели открытых в нижней части основного окна приложения (рис. 3.235).

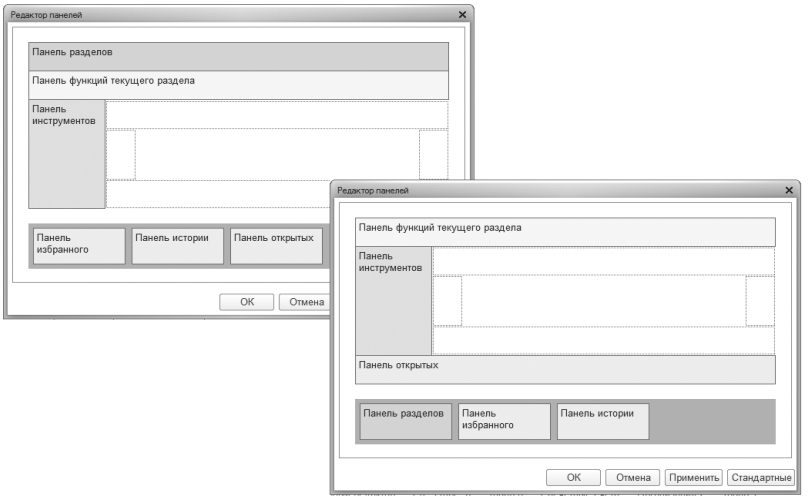


Рис. 3.235. Настройка панелей администратором в режиме «1С:Предприятие»

После изменения эти настройки были сохранены в системном хранилище для пользователя с ролью Администратор. И теперь при старте прикладного решения для администратора в основном окне приложения вместо стандартного состава и расположения панелей интерфейса показывается тот вид, который он настроил для себя в режиме 1С:Предприятие (рис. 3.236).

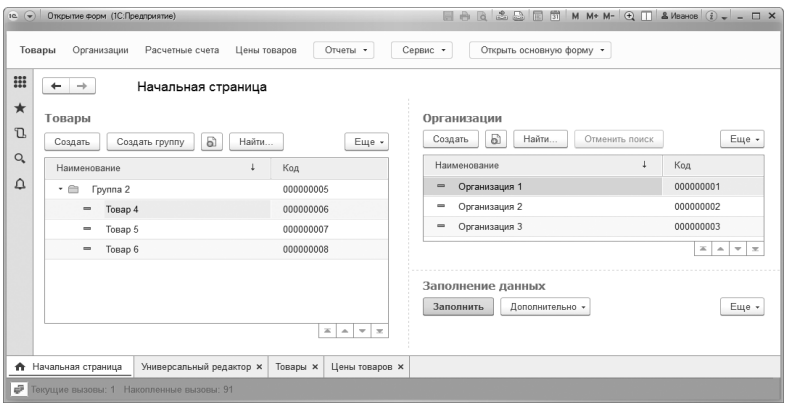


Рис. 3.236. Состав панелей интерфейса, настроенных администратором

Теперь мы хотим немного изменить и расширить эти настройки из встроенного языка. А именно – мы хотим отображать для администратора в левой части основного окна приложения панель избранного, а сверху, несмотря на его личные настройки, отображать панель инструментов и панель функций текущего раздела, причем не друг под другом, а в одной строке.

Для этого настройки пользователя надо прочитать из системного хранилища, изменить и записать обратно.

Напомним, что процедура `НастройкаИнтерфейса()` расположена в общем модуле РаботаСИнтерфейсом и вызывается из модуля управляемого приложения в обработчике `ПриНачалеРаботыСистемы()`. В процедуре настройки анализируются роли пользователей, и для пользователя с ролью Администратор вызывается процедура формирования расширенного интерфейса.

Поместим процедуру `РасширенныйИнтерфейс()` в том же общем модуле и заполним ее следующим образом (листинг 3.188).

Листинг 3.188. Процедура «РасширенныйИнтерфейс»

```
Процедура РасширенныйИнтерфейс() Экспорт

    НастройкиИнтерфейса = ХранилищеСистемныхНастроек.
        Загрузить("Общее/НастройкиИнтерфейсаКлиентскогоПриложения");
    НастройкиСостава = НастройкиИнтерфейса.ПолучитьСостав();

    Лево = НастройкиСостава.Лево;
    Верх = НастройкиСостава.Верх;
    Низ = НастройкиСостава.Низ;
    Право = НастройкиСостава.Право;

    ПанельИнструментов = Новый ЭлементНастройкиСоставаИнтерфейсаКлиентскогоПриложения(
            "ПанельИнструментов");
    ПанельФункцийТекущегоРаздела = Новый
            ЭлементНастройкиСоставаИнтерфейсаКлиентскогоПриложения(
            "ПанельФункцийТекущегоРаздела");
    ПанельИзбранного = Новый ЭлементНастройкиСоставаИнтерфейсаКлиентскогоПриложения(
            "ПанельИзбранного");

    Если Верх.Количество() > 0 Тогда
        Верх.Очистить();
    КонецЕсли;

    ГруппаВерх = Новый НастройкиСоставаИнтерфейсаКлиентскогоПриложения();
    ГруппаВерх.Верх.Добавить(ПанельИнструментов);
```

```
        ГруппаВерх.Верх.Добавить(ПанельФункцийТекущегоРаздела);
        Верх.Добавить(ГруппаВерх.Верх);

        Для Каждого Панель Из Лево Цикл
            Если Панель.Имя = "ПанельИнструментов" Тогда
                Лево.Удалить(Лево.Индекс(Панель));
            КонецЕсли;
            Если Панель.Имя = "ПанельФункцийТекущегоРаздела" Тогда
                Лево.Удалить(Лево.Индекс(Панель));
            КонецЕсли;
            Если Панель.Имя = "ПанельИзбранного" Тогда
                Лево.Удалить(Лево.Индекс(Панель));
            КонецЕсли;
        КонецЦикла;

        Лево.Добавить(ПанельИзбранного);

        НастройкиИнтерфейса.УстановитьСостав(НастройкиСостава);
        ХранилищеСистемныхНастроек.Сохранить(
            "Общее/НастройкиИнтерфейсаКлиентскогоПриложения",, НастройкиИнтерфейса);

КонецПроцедуры
```

В этой процедуре сначала мы загружаем из системного хранилища настройки текущего пользователя в объект `НастройкиИнтерфейсаКлиентскогоПриложения`. Затем с помощью метода этого объекта `ПолучитьСостав()` мы получаем настройки состава интерфейса, заданные пользователем. После этого с помощью свойств `Верх`, `Лево`, `Право`, `Низ` получаем список панелей и групп панелей, размещенных в верхней, левой, правой и нижней частях основного окна клиентского приложения.

Затем создаем конструктором новые элементы настройки состава интерфейса на основании стандартных имен панелей: `ПанельИнструментов`, `ПанельФункцийТекущегоРаздела` и `ПанельИзбранного`.

Нужно обратить внимание на один момент. Здесь, в отличие от настройки панелей для менеджера, мы не очищаем списки панелей, так как нам нужно добавить свои панели к списку панелей пользователя.

Поэтому в общем случае, перед тем как добавлять панели, надо проверить, не содержатся ли они в каждой из четырех групп (`Верх`, `Лево`, `Право`, `Низ`) настроек состава интерфейса, заданных пользователем. Это необходимо сделать потому, что панель интерфейса

не может быть дважды включена ни в одну и ту же, ни в разные группы настроек состава интерфейса.

Для упрощения примера мы будем проверять наличие трех панелей (которые мы собираемся добавлять) только в левой группе настроек состава пользовательского интерфейса. И если наши панели там уже содержатся, будем их просто удалять. А верхнюю группу будем очищать, несмотря на настройки пользователя.

После выполнения этих проверок мы конструктором создаем пустой объект `НастройкиСоставаИнтерфейсаКлиентскогоПриложения`. И добавляем в верхнюю часть этих настроек (`ГруппаВерх.Верх`) созданные ранее элементы настройки состава интерфейса – `ПанельИнструментов` и `ПанельФункцийТекущегоРаздела`. И затем уже эту новую группу настроек (`ГруппаВерх.Верх`) добавляем в верхнюю группу пользовательских настроек состава. Таким образом, мы расположили обе панели интерфейса в новой группе настроек и поместили эту группу вверху основного окна приложения пользователя.

`ПанельИзбранного` просто добавляем в левую группу пользовательских настроек состава, так как перед этим мы удаляем ее из списка панелей (если она там содержится).

После этого с помощью метода `УстановитьСостав()` мы загружаем измененные настройки состава в пользовательские настройки состава интерфейса.

В заключение сохраняем эти настройки для текущего пользователя в хранилище системных настроек с ключом `Общее/НастройкиИнтерфейсаКлиентскогоПриложения`.

Запустим «1С:Предприятие» от имени пользователя с ролью `Администратор`. Мы видим, что в левой части основного окна приложения появилась панель избранного, а сверху, несмотря на личные настройки пользователя, в одной строке располагаются панель инструментов и панель функций текущего раздела. При этом панель открытых отображается в нижней части основного окна приложения – в соответствии с пользовательскими настройками, сделанными ранее (рис. 3.237).

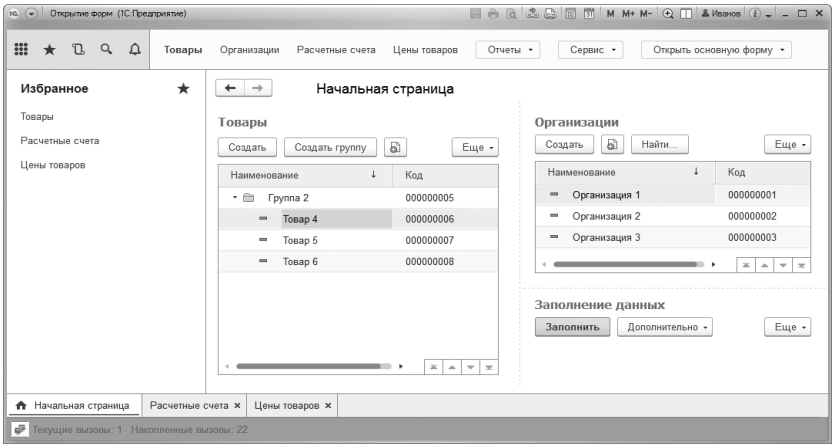


Рис. 3.237. Расширенный состав панелей интерфейса для администратора

Настройка состава форм на начальной странице

По аналогии с предыдущим примером можно при старте прикладного решения в зависимости от роли пользователя программно формировать его начальную страницу.

Основной объект, с помощью которого выполняется настройка начальной страницы, – НастройкиНачальнойСтраницы. Он позволяет работать как с настройками, заданными в конфигураторе разработчиком, так и с настройками, которые пользователь установил для себя в режиме 1С:Предприятие.

Продолжим наш пример. В процедуру НастройкаИнтерфейса() общего модуля РаботаСИнтерфейсом добавим вызов процедур для формирования начальной страницы для пользователя с ролью Администратор и для пользователя с ролью Менеджер (листинг 3.189).

Листинг 3.189. Процедура «НастройкаИнтерфейса»

```
Процедура НастройкаИнтерфейса() Экспорт

    ПользовательИБ = ПользователиИнформационнойБазы.ТекущийПользователь();
    Для Каждого РольПользователя Из ПользовательИБ.Роли Цикл
        Если РольПользователя.Имя = "Администратор" Тогда
            РасширенныйИнтерфейс();
            СложнаяНачальнаяСтраница();
```

```
            КонецЕсли;
            Если РольПользователя.Имя = "Менеджер" Тогда
                МинимальныйИнтерфейс();
                ПростаяНачальнаяСтраница();
            КонецЕсли;
        КонецЦикла;

КонецПроцедуры
```

На начальной странице менеджера мы будем отображать слева форму списка справочника Товары, а справа – форму отчета ЦеныТоваров. Причем независимо от его личных настроек и настроек, сделанных в конфигураторе.

Поместим в общем модуле РаботаСИнтерфейсом процедуру ПростаяНачальнаяСтраница() и заполним ее следующим образом (листинг 3.190).

Листинг 3.190. Процедура «ПростаяНачальнаяСтраница»

```
Процедура ПростаяНачальнаяСтраница() Экспорт

    НачальнаяСтраница = Новый НастройкиНачальнойСтраницы;
    СоставФорм = НачальнаяСтраница.ПолучитьСоставФорм();

    СоставФорм.ЛеваяКолонка.Очистить();
    СоставФорм.ПраваяКолонка.Очистить();

    СоставФорм.ЛеваяКолонка.Добавить("Справочник.Товары.ФормаСписка");
    СоставФорм.ПраваяКолонка.Добавить("Отчет.ЦеныТоваров.Форма.ФормаОтчета");

    НачальнаяСтраница.УстановитьСоставФорм(СоставФорм);
    ХранилищеСистемныхНастроек.Сохранить(
        "Общее/НастройкиНачальнойСтраницы", "", НачальнаяСтраница);

КонецПроцедуры
```

В этой процедуре мы конструктором создаем пустой объект НастройкиНачальнойСтраницы. Затем с помощью метода этого объекта ПолучитьСоставФорм() мы получаем состав форм начальной страницы, заданный в конфигураторе.

В нашем случае в конфигураторе начальная страница настроена следующим образом (рис. 3.238).

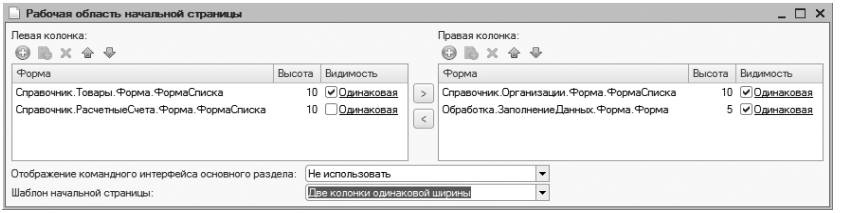


Рис. 3.238. Состав форм на начальной странице, заданный в конфигураторе

Очищаем список форм, размещенных в левой и правой колонке начальной страницы. И добавляем в левую колонку форму списка справочника Товары, а в правую – форму отчета ЦеныТоваров.

При указании имени формы можно использовать как полные имена форм объектов конфигурации (например, `Справочник.Товары.Форма.ИмяФормы`), так и стандартные имена форм (например, `Справочник.Товары.ФормаСписка`).

Также нужно учитывать, что при программной настройке начальной страницы нельзя изменить шаблон начальной страницы. Будет использоваться тот шаблон, который задан в конфигураторе.

После этого с помощью метода `УстановитьСоставФорм()` мы загружаем измененные настройки начальной страницы в пустой объект `НастройкиНачальнойСтраницы`.

И в заключение сохраняем эти настройки для текущего пользователя в хранилище системных настроек с ключом `Общее/НастройкиНачальнойСтраницы`.

Запустим «1С:Предприятие» от имени пользователя с ролью Менеджер. Независимо от настроек, сделанных в конфигураторе, и от личных настроек менеджера, сделанных в прошлом сеансе работы, при старте прикладного решения на его начальной странице всегда слева будет отображаться список товаров, а справа – отчет, показывающий цены товаров (рис. 3.239).

Теперь рассмотрим формирование начальной страницы для пользователя с ролью Администратор на основе настроек, сделанных им в режиме 1С:Предприятие.

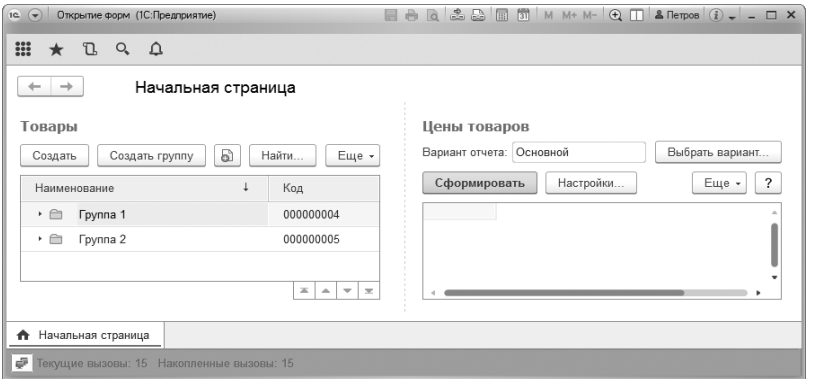


Рис. 3.239. Начальная страница приложения для менеджера

Предположим, администратор в пользовательском режиме изменил состав форм на начальной странице, заданный в конфигураторе. Он скрыл форму списка товаров, перенес из правой колонки начальной страницы в левую колонку форму списка справочника Организации и форму обработки Заполнение данных. А также администратор перетащил форму списка справочника Расчетные счета из списка доступных форм в правую колонку начальной страницы (рис. 3.240).

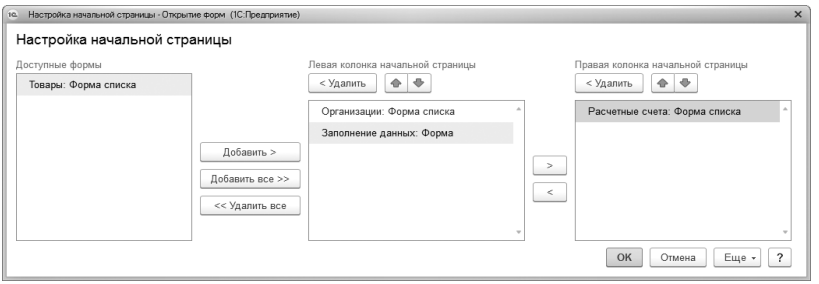


Рис. 3.240. Настройка начальной страницы администратором в режиме «1С:Предприятие»

После изменения эти настройки были сохранены в системном хранилище для пользователя с ролью Администратор. И теперь при старте прикладного решения на начальной странице администратора показывается тот вид, который он настроил для себя в режиме 1С:Предприятие (рис. 3.241).

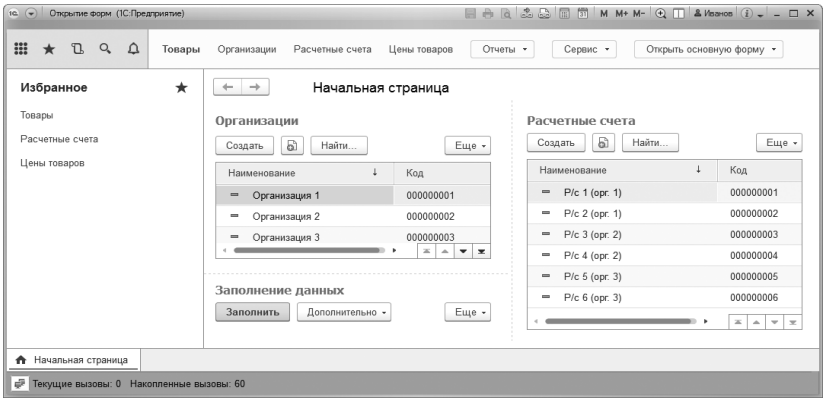


Рис. 3.241. Начальная страница, настроенная администратором

Теперь мы хотим немного изменить и расширить эти настройки из встроенного языка. А именно – мы хотим отображать для администратора в левой колонке начальной страницы форму списка справочника Организации, а в правой – форму обработки Заполнение данных. Остальные настройки, которые администратор сделал в пользовательском режиме, должны остаться без изменения.

Для этого настройки пользователя надо прочитать из системного хранилища, изменить и записать обратно.

Поместим в общем модуле РаботаСИнтерфейсом процедуру СложнаяНачальнаяСтраница() и заполним ее следующим образом (листинг 3.191).

Листинг 3.191. Процедура «СложнаяНачальнаяСтраница»

```
Процедура СложнаяНачальнаяСтраница() Экспорт

    НастройкиНачальнойСтраницы = ХранилищеСистемныхНастроек.Загрузить(
        "Общее/НастройкиНачальнойСтраницы");
    СоставФорм = НастройкиНачальнойСтраницы.ПолучитьСоставФорм();

    Левая = СоставФорм.ЛеваяКолонка;
    Правая = СоставФорм.ПраваяКолонка;

    ИмяФормы = "Справочник.Организации.Форма.ФормаСписка";
    ИндексФормы = Правая.Найти(ИмяФормы);
    Если ИндексФормы <> Неопределено Тогда
        Правая.Удалить(ИндексФормы);
```

```
        КонецЕсли;
        ИндексФормы = Левая.Найти(ИмяФормы);
        Если ИндексФормы = Неопределено Тогда
            Левая.Добавить("Справочник.Организации.Форма.ФормаСписка");
        КонецЕсли;

        ИмяФормы = "Обработка.ЗаполнениеДанных.Форма.Форма";
        ИндексФормы = Левая.Найти(ИмяФормы);
        Если ИндексФормы <> Неопределено Тогда
            Левая.Удалить(ИндексФормы);
        КонецЕсли;
        ИндексФормы = Правая.Найти(ИмяФормы);
        Если ИндексФормы = Неопределено Тогда
            Правая.Добавить("Обработка.ЗаполнениеДанных.Форма.Форма");
        КонецЕсли;

        НастройкиНачальнойСтраницы.УстановитьСоставФорм(СоставФорм);
        ХранилищеСистемныхНастроек.Сохранить(
            "Общее/НастройкиНачальнойСтраницы", "", НастройкиНачальнойСтраницы);

КонецПроцедуры
```

В этой процедуре сначала мы загружаем из системного хранилища настройки текущего пользователя в объект `НастройкиНачальнойСтраницы`. Затем с помощью метода этого объекта `ПолучитьСоставФорм()` мы получаем состав форм начальной страницы, заданный пользователем. После этого с помощью свойств `ЛеваяКолонка` и `ПраваяКолонка` получаем список форм, размещенных в левой и правой колонках начальной страницы.

Здесь, в отличие от настройки начальной страницы для менеджера, мы не очищаем списки форм, так как нам нужно добавить свои формы к списку форм пользователя.

Надо заметить, что форма может несколько раз содержаться и в одной и той же, и в разных колонках начальной страницы. Но это будет по меньшей мере «некрасиво» и совершенно не нужно.

Поэтому перед добавлением форм на начальную страницу мы будем проверять их наличие в настройках пользователя. И будем добавлять форму в случае ее отсутствия той колонке, в которую мы хотим ее добавить, и удалять форму, если она уже присутствует в другой колонке.

Надо учитывать, что для поиска в списках форм левой и правой колонки нужно использовать полные имена форм, соответственно, и при добавлении форм на начальную страницу нужно также использовать эти имена.

После этого с помощью метода `УстановитьСоставФорм()` мы загружаем измененные настройки в пользовательские настройки начальной страницы.

В заключение сохраняем эти настройки для текущего пользователя в хранилище системных настроек с ключом `Общее/НастройкиНачальнойСтраницы`.

Запустим «1С:Предприятие» от имени пользователя с ролью Администратор. Мы видим, что на начальной странице администратора слева располагается форма списка справочника Организации, а справа – форма обработки Заполнение данных. Но над ней в правой колонке начальной страницы располагается форма списка справочника Расчетные счета – в соответствии с пользовательскими настройками, сделанными ранее (рис. 3.242).

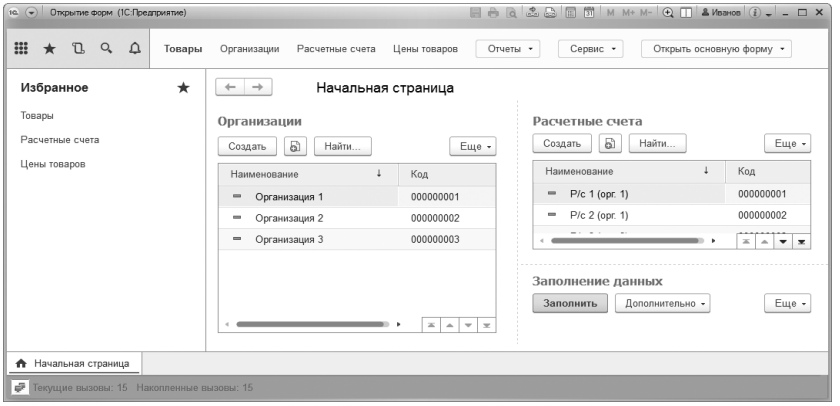


Рис. 3.242. Начальная страница приложения для администратора

ЧАСТЬ 4

Оптимизация клиент-серверного взаимодействия в формах

Оглавление

Глава 4.1. Общие рекомендации по оптимизации клиент-серверного взаимодействия

Форма должна быть «легкой», доступной в веб-клиенте и работать максимально быстро даже при использовании низкоскоростных каналов связи. Данные формы присутствуют как на клиенте, так и на сервере и автоматически синхронизируются при клиент-серверном взаимодействии.

Передача данных между сервером и клиентом происходит во многих случаях – например, при открытии формы и записи данных платформа сама преобразовывает прикладные объекты в данные формы и обратно. При контекстных вызовах процедур модуля формы все изменения данных формы и ее элементов платформа передает на сервер и обратно, и т. д.

Однако нужно понимать, что любое обращение на сервер – это непростой процесс. Платформа формирует обращение к серверу, передает его по каналу связи, выполняет какие-то действия на сервере, возвращает ответ по каналу связи… При работе тонкого клиента через GPRS каждый вызов сервера занимает примерно 1,5 секунды. Это, естественно, сказывается на быстродействии прикладного решения.

Поэтому *разработку прикладного решения необходимо вести с контролем количества вызовов серверных процедур и функций из клиентского кода, а также объема передаваемых данных между клиентом и сервером (трафика).*

Общее количество серверных вызовов складывается:

- из обращений на сервер, которые выполняет платформа «1С:Предприятие»;
- вызовов, которые выполняются из клиентского кода конфигурации разработчиком.

Если рассматривать источник этих вызовов, то существует несколько категорий серверных вызовов:

- Вызовы, которые платформа делает в результате того, что пользователь совершает какие-то интерактивные действия. На это разработчик повлиять не может.
- Вызовы, которые платформа также делает самостоятельно, но при выполнении определенных методов или свойств программного кода. На это разработчик может повлиять, если попробует использовать другие методы, свойства или изменить алгоритм.
- Серверные вызовы, которые разработчик делает в явном виде в коде.

Таким образом, разработчику при проектировании клиент-серверного взаимодействия в конфигурации нужно стремиться минимизировать количество вызовов сервера, которые он инициирует своим кодом.

Также разработчику нужно оптимизировать не только количество вызовов, но и объем передаваемых данных между клиентом и сервером (трафик). Неэффективность решения особенно будет заметна при работе через низкоскоростные каналы связи. Чтобы визуально оценить скорость работы на плохом канале связи, рекомендуется запускать клиентское приложение в режиме низкой скорости соединения, а в параметрах системы включать режим имитации задержек при вызове сервера.

С другой стороны, нужно минимизировать и объем кода, выполняющегося на клиенте. Это требование продиктовано:

- тем, что, как правило, клиентский компьютер менее производительный, чем серверный компьютер;
- необходимостью приемлемого качества работы в веб-клиенте. Клиентский код выполняется интерпретатором встроенного языка, который в веб-клиенте работает медленнее, чем в тонком или толстом клиенте.

В связи с этим методика оптимизации клиент-серверного взаимодействия базируется на следующих общих рекомендациях:

- Клиент и сервер рассматриваются как два взаимодействующих приложения.

- Разработчик программирует в явном виде серверную и клиентскую части конфигурации.
- Структура кода определяется логикой клиент-серверного взаимодействия, а не логикой того прикладного алгоритма, который реализует разработчик.
- Клиентский код продумывается не как последовательность действий, которую нужно выполнить, а как сценарий передачи управления между клиентом и сервером.
- Разработчик должен минимизировать частоту вызовов сервера. В идеале одному действию пользователя должен соответствовать один вызов сервера. Например, «хорошим тоном» разработки считается открытие формы за один серверный вызов. Конечно, эту рекомендацию нельзя считать строго обязательной, но желательно к этому стремиться.
- Разработчик должен пытаться сократить трафик – объем передаваемой между клиентом и сервером информации.
- Нужно реализовывать алгоритмы так, чтобы было минимум кода на клиенте, но и минимум обращений к серверу.

Важно понимать, что стремление оптимизировать код конфигурации должно иметь разумные пределы. Не нужно слепо следовать инструкциям и впадать в крайности. В погоне за оптимизацией можно сделать программный код совершенно нечитаемым. То есть за приемами оптимизации уже не будет видно собственно самой прикладной задачи, которая реализуется этим кодом.

В этом случае поддержка такого кода станет очень сложным делом. Человек, который не принимал участия в написании этого кода, скорее всего, просто не станет раскапывать этот кладезь оптимизации. А сам разработчик через полгода – год тоже с большим трудом сможет разобраться в том, что же он написал…

Таким образом, *нужно искать золотую середину между стройным читаемым кодом и его оптимизацией. Не нужно всегда бездумно стремиться только к одной лишь оптимизации.*

Глава 4.2. Инструменты, используемые при оптимизации клиент-серверного взаимодействия

В процессе отладки и работы конфигурации разработчику доступны следующие инструменты, которые он может использовать при оптимизации клиент-серверного взаимодействия:

- Показатели производительности.
- Режим низкой скорости соединения.
- Режим имитации задержек при вызове сервера.
- Отображение серверных вызовов в замерах производительности.
- Проверка серверных вызовов в обработчиках событий.

Рассмотрим каждый из них подробнее.

Показатели производительности

Показатели производительности прикладного решения позволяют разработчику оценить, насколько эффективно работает прикладное решение. Они дают информацию о количестве текущих и накопленных вызовов сервера, длительности вызовов сервера, объеме принятых и отправленных данных (рис. 4.1).

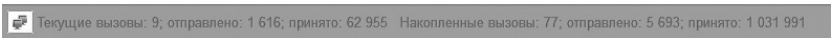


Рис. 4.1. Показатели производительности в информационной панели окна «1С:Предприятия»

Текущие вызовы – это вызовы сервера с момента последнего действия пользователя. *Накопленные вызовы* – это вызовы сервера с момента запуска приложения или с момента обнуления накопленных показателей.

Принято – это объем данных, принятых клиентом от сервера. *Отправлено* – это объем данных, отправленных с клиента на сервер.

Запустив «1С:Предприятие» при включенном отображении показателей производительности, разработчик в реальном времени может увидеть и оценить количество и длительность серверных вызовов,

а также объем переданных данных между клиентом и сервером. Анализ этих показателей позволит разработчику оптимизировать клиент-серверное взаимодействие в целях повышения эффективности работы прикладного решения.

Показатели производительности системы отображаются в информационной панели, расположенной в нижней части основного окна приложения. А также там отображается иконка, показывающая, включен ли режим имитации задержек при вызове сервера.

Значок  в информационной панели приложения означает, что отображение показателей производительности включено. Включить/выключить его можно как в настройках конфигуратора, так и в режиме 1С:Предприятие.

При запуске сеанса «1С:Предприятия» из конфигуратора отображение показателей производительности по умолчанию включено. В этом можно убедиться, выполнив команду главного меню конфигуратора Сервис – Параметры – Запуск 1С:Предприятия – Дополнительные (рис. 4.2).

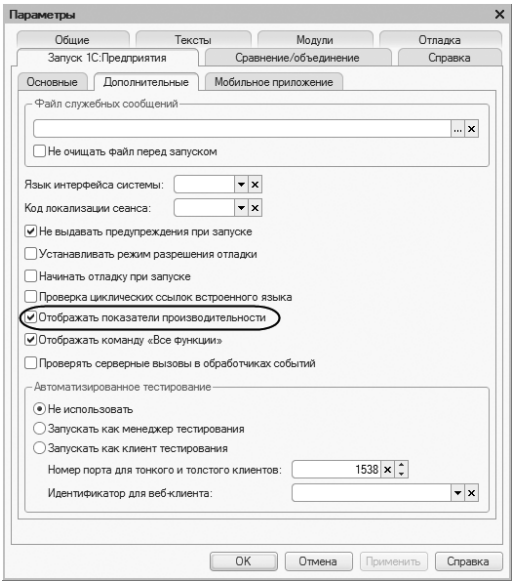


Рис. 4.2. Настройка отображения показателей производительности в режиме «Конфигуратор»

В текущем сеансе «1С:Предприятия» можно включить отображение показателей производительности, выполнив команду главного меню Сервис – Параметры и установив флажок Отображать показатели производительности (рис. 4.3).

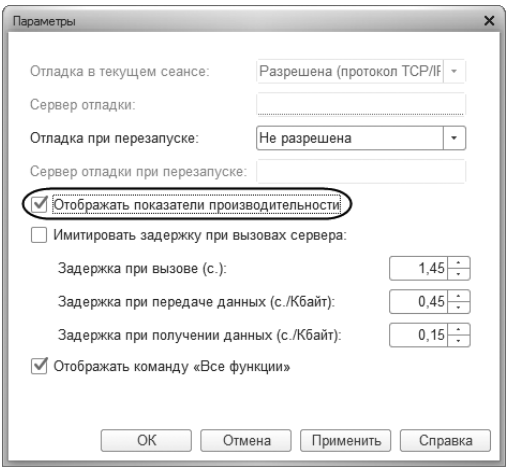


Рис. 4.3. Настройка отображения показателей производительности в режиме «1С:Предприятие»

Отображение показателей производительности также можно включить, запустив «1С:Предприятие» из командной строки с параметром `/DisplayPerformance`.

Стандартно в информационной панели будет показано только количество вызовов сервера (накопленное и текущее), но отображение других показателей производительности можно настроить в специальном окне.

Нажав на значок в информационной панели, можно выполнить различные команды контекстного меню: просмотреть историю текущих и накопленных вызовов сервера, очистить накопленные показатели.

Выбрав из контекстного меню пункт Настройка, можно настроить отображение различных показателей производительности: Объем отправленных данных, Объем принятых данных и т. д. (рис. 4.4).

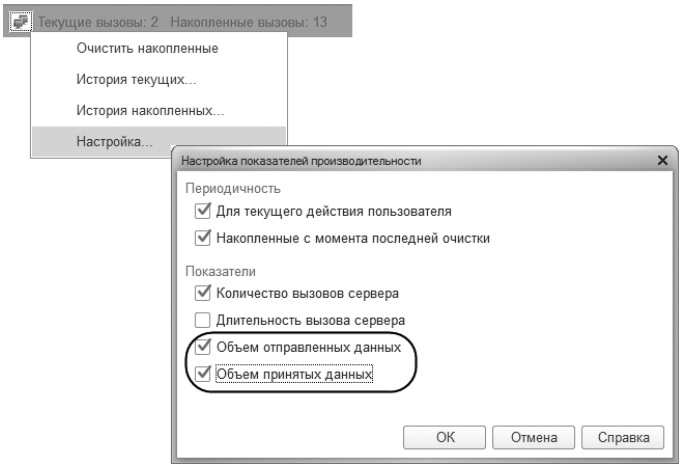


Рис. 4.4. Настройка состава показателей производительности

Таким образом, чтобы оптимизировать разрабатываемое приложение, разработчику нужно запустить «1С:Предприятие» в режиме отладки из конфигуратора и проанализировать показатели производительности, отражающие клиент-серверное взаимодействие в процессе работы прикладного решения.

Документация «1С:Предприятие 8.3.10. Руководство разработчика», глава 28.5 «Отладка и тестирование прикладных решений. Отображение вызовов сервера».

Режим низкой скорости соединения

При оптимизации клиент-серверного взаимодействия бывает важно и интересно посмотреть, насколько эффективно прикладное решение будет работать при использовании низкоскоростных каналов связи.

Дело в том, что клиентский компьютер может быть подключен к серверу как по локальной сети, так и через Интернет. Связь с сервером через Интернет может устанавливаться не только через быстрые оптоволоконные каналы, но и через модемы, использующие такие протоколы доступа к данным, как GPRS и т. п.

В этом случае администратору или пользователю «1С:Предприятия» нужно указать клиентскому приложению, что оно должно запуститься в режиме низкой скорости соединения.

Режим низкой скорости соединения – это способ работы клиентского приложения, позволяющий ему функционировать на медленных каналах связи как можно эффективнее.

При запуске в этом режиме платформа автоматически предпринимает шаги по оптимизации взаимодействия с сервером. Она отключает некоторую функциональность прикладного решения – например, не показывает начальную страницу и картинки разделов, объединяет и минимизирует вызовы сервера, дополнительно сжимает данные и т. д.

Таким образом, даже на плохом канале связи в режиме низкой скорости соединения «1С:Предприятие» будет обеспечивать приемлемую скорость работы пользователей.

Заметим, что установка низкой скорости соединения имеет смысл только для тонкого клиента и веб-клиента, так как толстый клиент не умеет работать через Интернет.

Задать низкую скорость соединения можно для всей информационной базы при ее создании или редактировании в списке информационных баз в стартовом окне «1С:Предприятия» (рис. 4.5).

Также если в свойстве Скорость соединения информационной базы установлено Выбирать при запуске, то режим низкой скорости соединения может задать сам пользователь в диалоге выбора информационной базы (рис. 4.6).

Кроме того, при любых настройках информационной базы низкую скорость соединения можно указать с помощью параметра командной строки.

Для запуска тонкого клиента используется параметр `/O Low` (листинг 4.1).

Листинг 4.1. Установка низкой скорости соединения тонкого клиента в командной строке

```
"C:\Program Files (x86)\1cv8\8.3.10.2505\bin\1cv8c.exe" /F D:\myBase /O Low
```

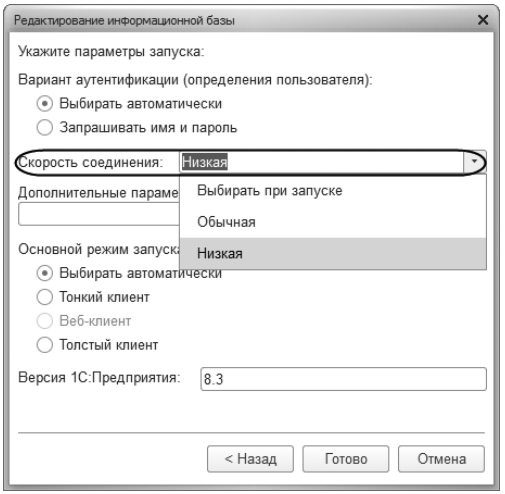


Рис. 4.5. Установка скорости соединения в свойствах информационной базы

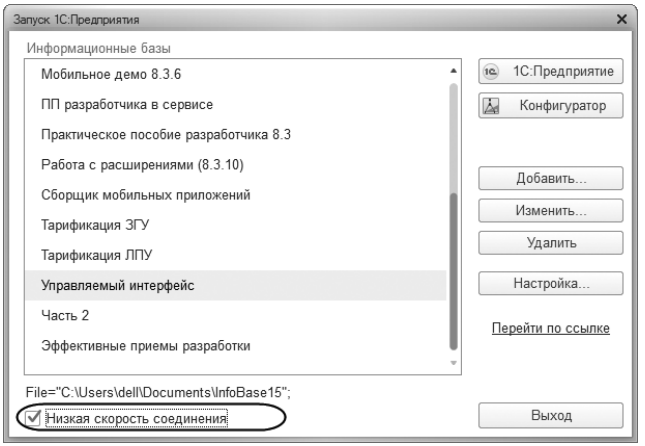


Рис. 4.6. Установка скорости соединения в диалоге выбора информационной базы

Для веб-клиента и тонкого клиента, подключаемого по протоколу HTTP, используется параметр `/O=Low` (листинг 4.2).

Листинг 4.2. Установка низкой скорости соединения веб-клиента в командной строке

```
http://demo-ma.1c.ru/demo_ma?N=Administrator&O=Low
```

Для сеансов «1С:Предприятия», запускаемых из конфигуратора, тоже можно задать режим низкой скорости соединения. Для этого нужно выполнить команду главного меню Сервис – Параметры – Запуск 1С:Предприятия – Основные и установить флажок Низкая скорость соединения (рис. 4.7).

Встроенная справка в режиме Конфигуратор – Справка – Содержание справки – Запуск 1С:Предприятия 8 и параметры запуска – Общие параметры запуска, а также Параметры запуска веб-клиента.

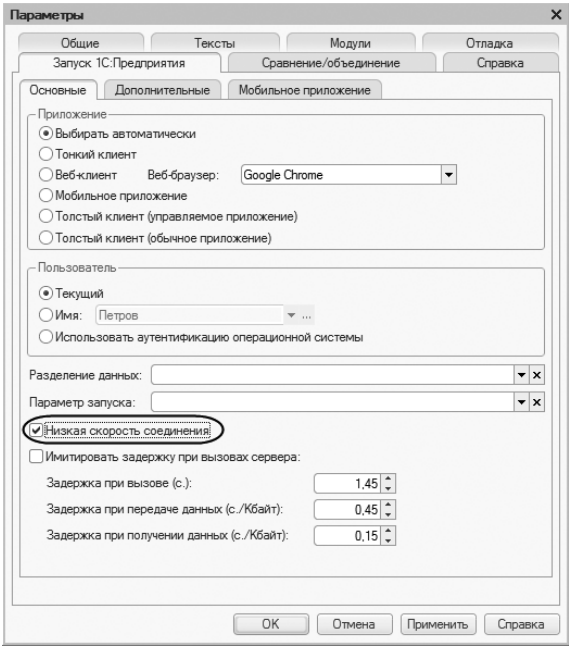


Рис. 4.7. Установка скорости соединения в режиме «Конфигуратор»

Режим низкой скорости соединения, как уже говорилось выше, сказывается на внешнем функционировании тонкого клиента и веб-клиента. Например, не отображается начальная страница при старте приложения, не отображаются картинки разделов, автоматически не формируется список выбора при вводе по строке и т. д.

Если в режиме 1С:Предприятие вызвать окно информации о программе (Справка – О программе), то можно увидеть, в какой скорости соединения запущено приложение (рис. 4.8).

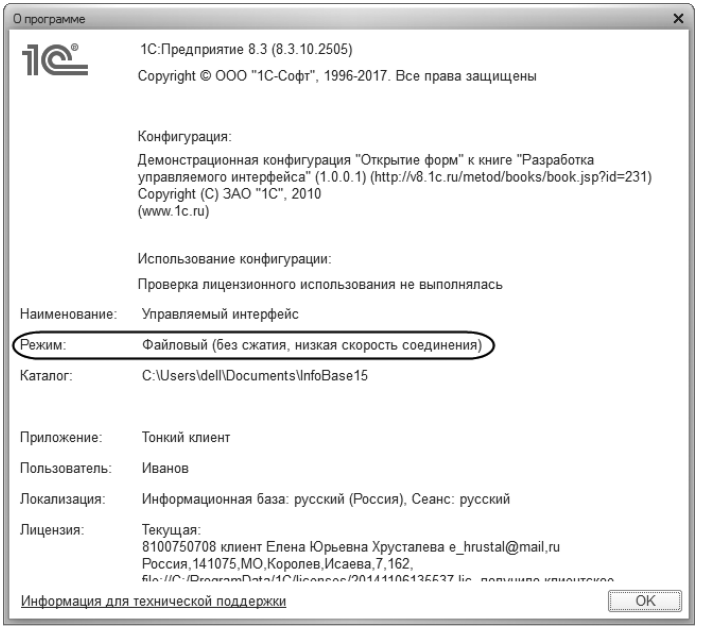


Рис. 4.8. Окно информации о программе

Заметим, что режим низкой скорости соединения задается один раз при запуске клиентского приложения и не может быть изменен в процессе его работы.

Таким образом, чтобы визуально оценить, с какой скоростью будет работать прикладное решение у клиента на низкоскоростном канале связи, разработчику нужно подключиться к Интернету, например через мобильный телефон, и запустить тонкий клиент в режиме низкой скорости соединения.

Документация «1С:Предприятие 8.3.10. Руководство разработчика», приложение 7.1 «Особенности поведения системы в различных режимах. Особенности режима низкой скорости соединения».

Имитация задержек при вызове сервера

Однако разработчик может и не использовать специальные медленные соединения через Интернет, но при этом оценить реальную скорость работы его прикладного решения у клиента. Для этого предназначен *режим имитации задержек при вызове сервера*. При включении этого режима платформа (даже в файловом варианте на локальном компьютере) будет работать с теми временными задержками, с которыми работает реальный канал связи.

Этот режим не нужно путать с режимом низкой скорости соединения.

Режим низкой скорости соединения – это, скорее, инструмент пользователя. Пользователь устанавливает этот режим, чтобы приложение на медленных каналах связи работало как можно быстрее. Данный режим задается при запуске клиентского приложения и не может быть изменен в процессе его работы.

Режим имитации задержек при вызове сервера – это инструмент разработчика, позволяющий ему превратить быстрый канал связи в медленный. Причем сделать это можно как при запуске приложения, так и в процессе его работы. При этом режим, в котором было запущено клиентское приложение, не изменится.

Заметим, что включить режим имитации задержек при вызове сервера можно только для тонкого клиента и толстого клиента, запущенного в режиме управляемого приложения.

Режим имитации задержек при вызове сервера можно задать в параметрах конфигуратора, чтобы при запуске в режиме отладки приложение сразу работало в «плохих условиях». Для этого нужно выполнить команду главного меню Сервис – Параметры – Запуск 1С:Предприятия – Основные и установить флажок Имитировать задержку при вызовах сервера (рис. 4.9).

При этом можно установить нужные временные задержки при вызове сервера и при передаче/получении данных с сервера. Платформа использует стандартные задержки, соответствующие мобильному интернет-соединению (через GPRS). Но разработчик может подобрать другие значения временных задержек, соответствующие характеристикам канала связи, используемого заказчиком.

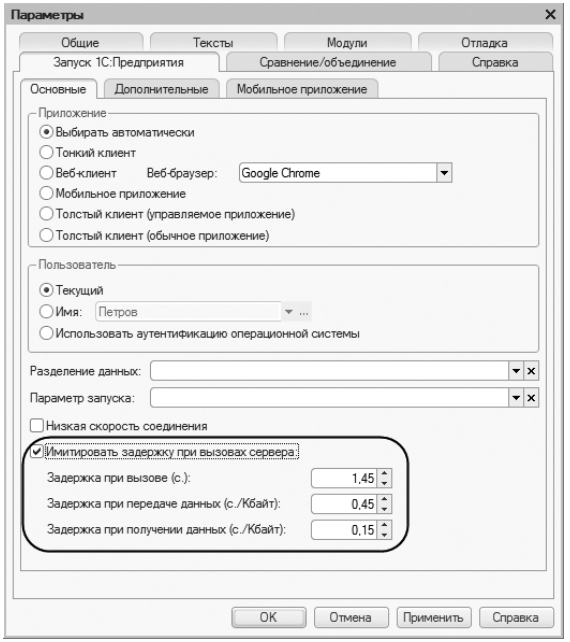


Рис. 4.9. Установка имитации задержек при вызове сервера в режиме «Конфигуратор»

При запуске «1С:Предприятия» в режиме имитации задержек при вызове сервера в информационной панели приложения появится значок .

В текущем рабочем сеансе также можно включить/выключить режим имитации задержек при вызове сервера. Для этого нужно выполнить команду главного меню «1С:Предприятия» Сервис – Параметры и установить флажок Имитировать задержку при вызовах сервера (рис. 4.10).

Режим имитации задержек при вызове сервера также можно включить, запустив «1С:Предприятие» из командной строки с параметром `/SimulateServerCallDelay [-CallXXXX] [-SendYYYY] [-ReceiveZZZZ]`.

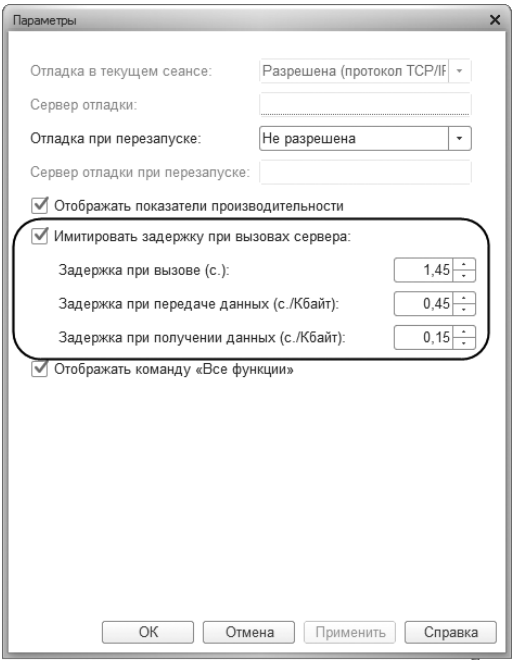


Рис. 4.10. Установка имитации задержек при вызове сервера в режиме «1С:Предприятие»

Таким образом, чтобы визуально оценить, с какой скоростью будет работать прикладное решение у клиента на низкоскоростном канале связи, разработчику нужно запустить тонкий клиент в режиме низкой скорости соединения и установить режим имитации задержек при вызове сервера (при этом не нужно использовать специальные медленные каналы связи, можно это сделать даже в файловом варианте на локальном компьютере).

Документация «1С:Предприятие 8.3.10. Руководство разработчика», глава 28.4 «Отладка и тестирование прикладных решений. Механизм имитации задержек вызове сервера».

Отображение серверных вызовов в замерах производительности

Замер производительности позволяет оценить скорость работы всей конфигурации или любой ее части в процессе отладки. При этом измеряются частота и скорость выполнения отдельных участков кода, указывается, где выполнялся код (на сервере или на клиенте), помечаются строки кода, приведшие к вызову сервера.

Для того чтобы выполнить замер производительности какого-либо участка конфигурации, нужно запустить «1С:Предприятие» из конфигуратора в режиме отладки, дойти в ходе выполнения приложения до интересующего участка, перейти в режим Конфигуратор и выполнить команду главного меню Отладка – Замер производительности. Затем вернуться в режим 1С:Предприятие, продолжить выполнение приложения, а когда замер производительности больше не нужен – повторно выполнить команду Отладка – Замер производительности из конфигуратора. После этого в конфигураторе откроется окно с результатами замера производительности, выполненного между этими двумя нажатиями (рис. 4.11).

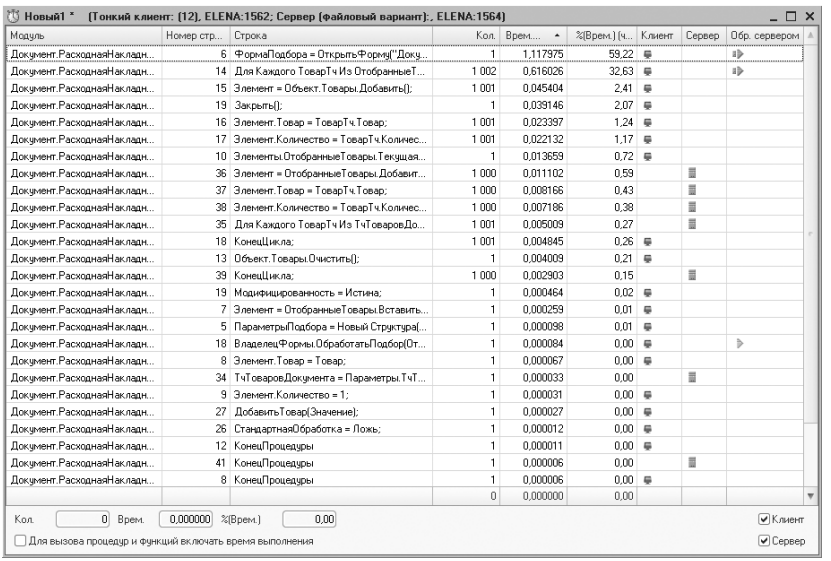


Рис. 4.11. Окно результатов замера производительности

Эти результаты можно сохранить в файл для их дальнейшего анализа с помощью команд Файл – Сохранить и Файл – Сохранить как. Файл результатов имеет расширение `*.pff`.

В окне результатов замера производительности в колонке Обр. сервером (Обработка сервером) показываются серверные вызовы, которые выполняются платформой и/или производятся из клиентского кода:

- вызовы сервера методами объектов встроенного языка, вызовы процедур и функций, исполнение которых происходит на сервере, отображаются с помощью значка ;
- вызовы клиентских процедур и функций, в которых тем или иным способом происходил вызов сервера, отображаются с помощью значка .

В окне результатов замера производительности показывается, где исполнялся код на встроенном языке в клиент-серверной информационной базе: на клиенте или на сервере:

- строки кода на встроенном языке, исполнение которых происходило на клиенте, отображаются в колонке Клиент с помощью значка ;
- строки кода на встроенном языке, исполнение которых происходило на сервере, отображаются в колонке Сервер с помощью значка .

По любой колонке окна результатов замера производительности возможна сортировка: для этого достаточно щелкнуть мышью в соответствующей колонке.

В окне результатов замера производительности существует возможность фильтрации информации результатов замера. Такая возможность реализована в виде двух флажков (Клиент и Сервер) в правой нижней части окна результатов замера. По умолчанию установлены оба флажка, то есть в результатах замера присутствует информация о ходе исполнения кода на встроенном языке как на клиенте, так и на сервере.

Документация «1С:Предприятие 8.3.10. Руководство разработчика», глава 28.3 «Отладка и тестирование прикладных решений. Замер производительности».

Анализ замеров производительности в процессе разработки может дать дополнительную информацию, детализирующую данные показателей производительности.

Совместное использование этих двух инструментов позволит выявить узкие места в приложении даже без наличия большой тестовой базы и без выполнения нагрузочного тестирования.

Проверка серверных вызовов в обработчиках событий

При оптимизации клиент-серверного взаимодействия разработчик должен учитывать, что в некоторых клиентских обработчиках событий формы (`ПередЗаписью()`, `ПослеЗаписи()`, `ПередЗакрытием()`, `ПриЗакрытии()` и др.) запрещается вызывать контекстные серверные процедуры.

При описании таких событий в синтакс-помощнике содержится специальное указание на запрещенное действие или описание действия, которое может привести к недопустимому вызову:

- В обработчике данного события нельзя использовать серверные методы формы с директивой компиляции `&НаСервере`.
- Изменение свойства на клиенте может потребовать обращения к серверу.
- Вызов метода выполняет обращение к серверу.

Последние два предупреждения означают, что использование указанных клиентских методов или изменение некоторых свойств на клиенте могут приводить к неявным серверным вызовам. Поэтому разработчик не может применять их в обработчике события, для которого запрещены контекстные серверные вызовы.

Дело в том, что подобные вызовы потенциально могут нарушить алгоритмы функционирования форм. Поэтому они запрещены на уровне платформы, в процессе исполнения прикладного кода в режиме 1С:Предприятие. Но в процессе написания кода разработчик может допустить такую ошибку. И для того, чтобы обнаружить такую ошибку заранее (а не в режиме исполнения), существует эта проверка.

При автоматическом создании клиентского обработчика события, в котором запрещены контекстные серверные вызовы, в окне выбора типа обработчика (на клиенте, на сервере без контекста или на сервере) последняя опция (вызов с клиента контекстной серверной процедуры) будет недоступна для выбора. В принципе, разработчик может проигнорировать это предупреждение и написать «руками» тот вызов, который ему требуется. Но, тогда ответственность за возможные ошибки в работе приложения будет лежать целиком на нем.

Диагностику нарушений вышеуказанных ограничений можно включить в параметрах конфигуратора. Для этого нужно выполнить команду главного меню Сервис – Параметры – Запуск 1С:Предприятия – Дополнительные и установить флажок Проверять серверные вызовы в обработчиках событий (рис. 4.12).

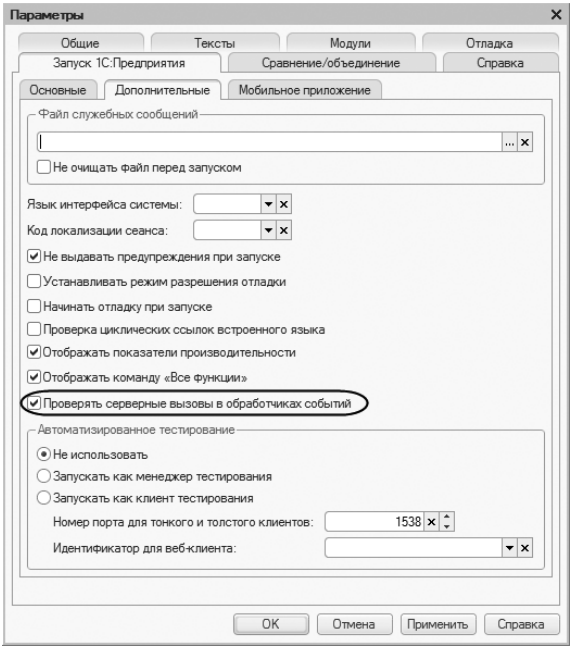


Рис. 4.12. Установка режима проверки серверных вызовов в обработчиках событий в режиме «Конфигуратор»

Режим проверки использования контекстных серверных вызовов формы в обработчиках событий также можно включить, запустив «1С:Предприятие» из командной строки с параметром `/EnableCheckServerCalls`.

При этом, если во время исполнения прикладного решения выполняется контекстный серверный вызов в обработчике события формы, в котором такие вызовы запрещены, в окно сообщений будет выведено диагностическое сообщение. Это же сообщение будет доступно в окне информации о программе по ссылке Информация для технической поддержки (рис. 4.13).

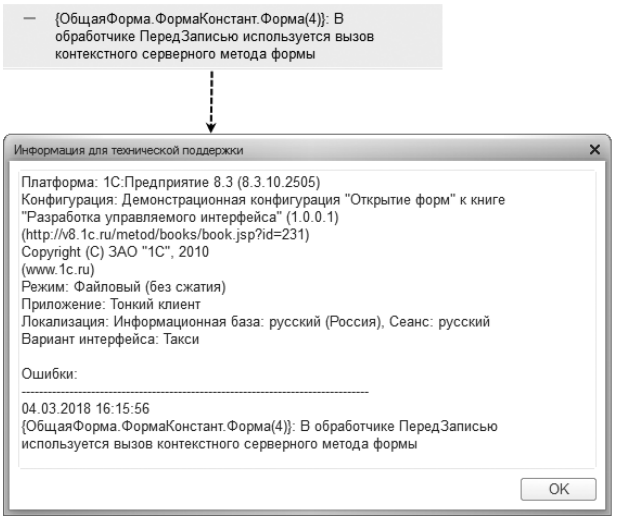


Рис. 4.13. Диагностика контекстных серверных вызовов в обработчиках событий формы в режиме «1С:Предприятие»

Документация «1С:Предприятие 8.3.10. Руководство разработчика», глава 4.8.3 «Встроенный язык. Особенности различных вариантов запуска системы. Особенности использования объектов, их свойств и методов», а также глава 33.2.5.2.

Глава 4.3. Примеры оптимизации клиент-серверного взаимодействия

Объединение нескольких вызовов сервера в один

В процессе работы формы часто бывает нужно получить дополнительную информацию об объекте, на который ссылается поле реквизита, и отразить ее в форме. Как это сделать наиболее эффективно? Рассмотрим пример.

Предположим, в конфигурации существует справочник Товары. В нем для каждого товара хранятся его артикул, единица измерения и цена. Для приходования товаров существует документ ПриходнаяНакладная, в табличную часть которого подбираются товары (рис. 4.14).

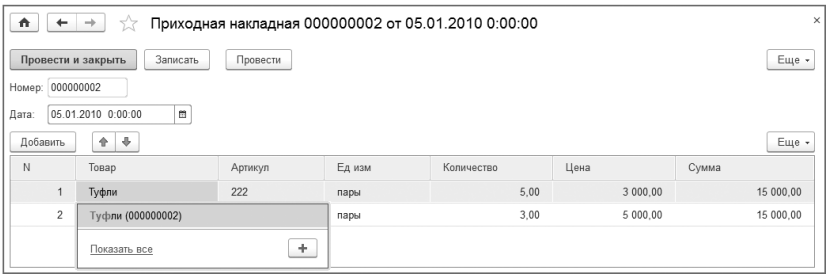


Рис. 4.14. Выбор товара в приходной накладной

Хочется, чтобы пользователь, выбрав товар, видел в документе не только ссылку, но сразу же и артикул, единицу измерения и цену выбранного товара (рис. 4.15).

Таким образом, в тот момент, когда пользователь выбирает товар в колонке Товар, ссылающейся на справочник Товары, в другие колонки табличной части документа Артикул, ЕдИзм и Цена нужно записать артикул, единицу измерения и цену этого товара, хранящиеся в справочнике.

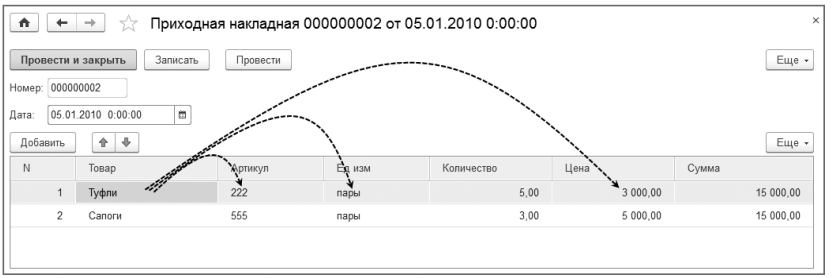


Рис. 4.15. Заполнение колонок табличной части в приходной накладной

Выбор товара происходит на клиенте, но на клиенте, имея ссылку на товар, нельзя получить значения реквизитов от этой ссылки, например Артикул.

Чтобы его получить, нужно «пойти» с этой ссылкой на сервер, там прочитать из базы данных значение артикула и передать его обратно на клиент.

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

Как уже говорилось выше, возможности ссылки на клиенте сильно ограничены. Чтобы воспользоваться полными возможностями ссылки, можно, например, создать универсальную функцию `ПолучитьРеквизитИзБазыДанных(Ссылка, ИмяРеквизита)`, поместить ее в общем модуле, исполняющемся на сервере, и использовать ее всякий раз, когда потребуется получить реквизит какого-нибудь объекта.

Поскольку функция универсальная, мы будем использовать ее в разных формах, поэтому поместим ее не в какой-то одной форме, а в общем модуле, чтобы все формы могли ею пользоваться.

Итак, создадим общий модуль конфигурации РаботаСОбъектами. По умолчанию у него установлен флажок Сервер, значит, экземпляры этого модуля будут скомпилированы на стороне сервера. Установим у него флажок Вызов сервера, чтобы экспортируемые процедуры и функции этого модуля можно было вызывать с клиента (рис. 4.16).

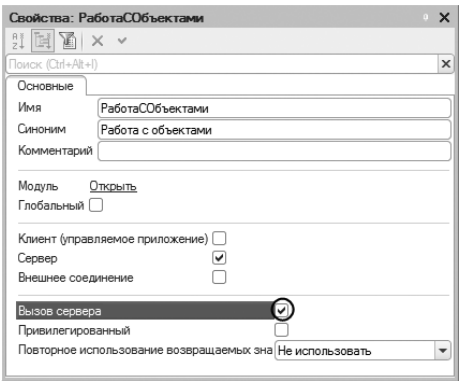


Рис. 4.16. Свойства модуля «РаботаСОбъектами»

Поместим в этом модуле функцию для получения реквизита объекта по его ссылке (листинг 4.3).

Листинг 4.3. Функция «ПолучитьРеквизитИзБазыДанных()»

```
Функция ПолучитьРеквизитИзБазыДанных(Ссылка, ИмяРеквизита) Экспорт

    Возврат Ссылка[ИмяРеквизита];

КонецФункции
```

Чтобы обеспечить заполнение колонок табличной части приходной накладной при выборе товара, создадим форму документа и обработчик события `ПриИзменении()` для поля формы `Товар`, содержащегося в таблице формы `Товары` (листинг 4.4).

Листинг 4.4. Процедура «ТоварыТоварПриИзменении()»

```
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

    ДанныеТекущейСтроки = Элементы.Товары.ТекущиеДанные;
    ВыбранныйТовар = ДанныеТекущейСтроки.Товар;
    ДанныеТекущейСтроки.Артикул = РаботаСОбъектами.
        ПолучитьРеквизитИзБазыДанных(ВыбранныйТовар, "Артикул");
    ДанныеТекущейСтроки.ЕдИзм = РаботаСОбъектами.
        ПолучитьРеквизитИзБазыДанных(ВыбранныйТовар, "ЕдиницаИзмерения");
    ДанныеТекущейСтроки.Цена = РаботаСОбъектами.
        ПолучитьРеквизитИзБазыДанных(ВыбранныйТовар, "Цена");

КонецПроцедуры
```

В этом обработчике мы получаем доступ к данным текущей строки таблицы формы в переменной `ДанныеТекущейСтроки`. Чтобы получить значение, содержащееся в конкретном поле текущей строки, нужно через точку указать имя этого поля (`.Товар`). Так мы получаем ссылку на выбранный товар и вместе с именем нужного реквизита передаем ее в функцию `ПолучитьРеквизитИзБазыДанных()`, которая возвращает нам значение этого реквизита.

> Этот пример можно посмотреть в демонстрационной конфигурации «01 (вар. 1) Объединение нескольких вызовов сервера в один».

Посмотрим теперь, какие серверные вызовы будут происходить в режиме 1С:Предприятие в результате.

Для этого запустим «1С:Предприятие», откроем приходную накладную и сделаем выбор из справочника товаров в колонке Товар. После этого колонки табличной части Артикул, ЕдИзм и Цена будут автоматически заполнены соответствующими значениями из справочника Товары (рис. 4.17).

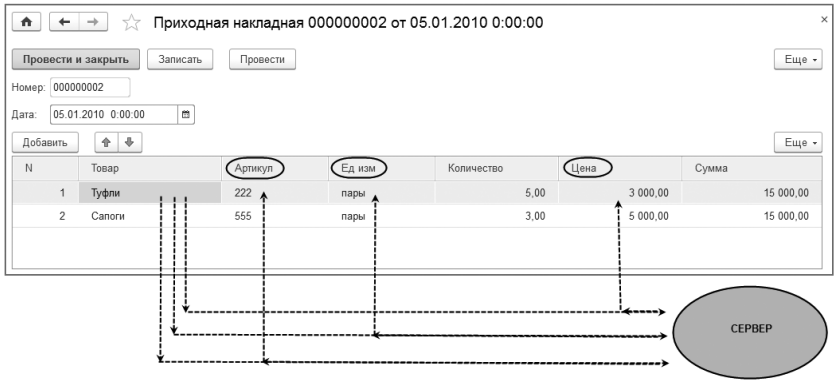


Рис. 4.17. Заполнение колонок табличной части в приходной накладной

Сразу после выбора товара посмотрим на показатели производительности (рис. 4.18).

Рис. 4.18. Показатели производительности

Там будет показано число вызовов сервера, в общем случае их будет три или больше. Больше трех вызовов может быть при первом выполнении этого действия, так как платформа выполняет запрос к информационной базе и кеширование необходимых данных. При втором и последующем выборе товара произойдет три вызова сервера.

Как мы уже говорили выше, когда платформа самостоятельно обращается к серверу, на это мы повлиять не можем. Поэтому подробно исследовать, почему серверных вызовов больше трех, мы не будем. Но три серверных вызова – это «наши вызовы», которые мы породили своим кодом. Рассмотрим подробнее, откуда они взялись.

Так происходит потому, что в результате использования универсальной функции `ПолучитьРеквизитИзБазыДанных()` будет происходить обращение на сервер столько раз, сколько понадобится получить реквизитов объекта. Таким образом, при выборе товара в строке табличной части документа произошло *три вызова сервера*, так как нам понадобилось узнать значение трех реквизитов выбранного товара.

Второй вариант решения

На самом деле нет необходимости три раза «дергать» сервер. Ведь мы точно знаем, что каждый раз при добавлении нового товара нам понадобятся все три его реквизита. А значит, все их можно получить за один серверный вызов. Это будет более эффективным решением.

В данном случае универсальность выбранного нами решения нужно принести в жертву эффективности работы конкретной формы в конкретной ситуации. В данной ситуации прежде всего нас должна заботить не универсальность, а прикладная логика работы формы. Универсальность сама по себе хороша, но не в данном случае. То есть нет одного решения на все времена. В каждой форме нужно думать заново.

Для реализации этого решения в обработчике события ПриИзменении() поля Товар табличной части документа ПриходнаяНакладная нужно вызвать функцию, возвращающую в виде структуры сразу все нужные нам реквизиты выбранного элемента. Поэтому в модуле формы документа поместим функцию ПолучитьРеквизитыТовара(), выполняющуюся на сервере без контекста формы (листинг 4.5).

Листинг 4.5. Функция «ПолучитьРеквизитыТовара()»

```
&НаСервереБезКонтекста
Функция ПолучитьРеквизитыТовара(Ссылка)

	РеквизитыТовара = Новый Структура;
	РеквизитыТовара.Вставить("Артикул", Ссылка.Артикул);
	РеквизитыТовара.Вставить("ЕдиницаИзмерения", Ссылка.ЕдиницаИзмерения);
	РеквизитыТовара.Вставить("Цена", Ссылка.Цена);

	Возврат РеквизитыТовара;

КонецФункции
```

В этой функции мы создаем структуру и заполняем ее поля Артикул, ЕдиницаИзмерения и Цена значениями соответствующих реквизитов справочника Товары, полученных по ссылке на элемент справочника.

Единственное значение данных формы, которое нам понадобится в этой функции, – это ссылка на элемент справочника, его мы передаем в параметре Ссылка. Поэтому мы выполняем внеконтекстный серверный вызов (директива компиляции &НаСервереБезКонтекста), который будет работать значительно быстрее, чем вызов сервера с контекстом формы (директива компиляции &НаСервере). Подробнее этот вопрос будет рассмотрен в следующем примере.

Как видите, функция будет совершенно не универсальная, зато форма будет работать быстро за счет *минимизации серверных вызовов*. То есть вместо трех вызовов сервера, как раньше, будет происходить всего один серверный вызов.

Теперь изменим обработчик события ПриИзменении() поля Товар табличной части документа ПриходнаяНакладная следующим образом (листинг 4.6).

Листинг 4.6. Процедура «ТоварыТоварПриИзменении()»

```
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

    ДанныеТекущейСтроки = Элементы.Товары.ТекущиеДанные;
    ВыбранныйТовар = ДанныеТекущейСтроки.Товар;
    РеквизитыТовара = ПолучитьРеквизитыТовара(ВыбранныйТовар);
    ДанныеТекущейСтроки.Артикул = РеквизитыТовара.Артикул;
    ДанныеТекущейСтроки.ЕдИзм = РеквизитыТовара.ЕдиницаИзмерения;
    ДанныеТекущейСтроки.Цена = РеквизитыТовара.Цена;

КонецПроцедуры
```

В этом обработчике мы передаем ссылку на выбранный товар в функцию `ПолучитьРеквизитыТовара()`, которая возвращает в виде структуры реквизиты товара, и присваиваем значение полей возвращенной структуры колонкам табличной части Артикул, ЕдИзм и Цена.

Этот пример можно посмотреть в демонстрационной конфигурации «01 (вар. 2) Объединение нескольких вызовов сервера в один».

Запустим «1С:Предприятие», откроем приходную накладную и сделаем выбор из справочника товаров в колонке Товар табличной части документа. После этого колонки табличной части Артикул, ЕдИзм и Цена будут автоматически заполнены соответствующими значениями из справочника Товары (рис. 4.19).

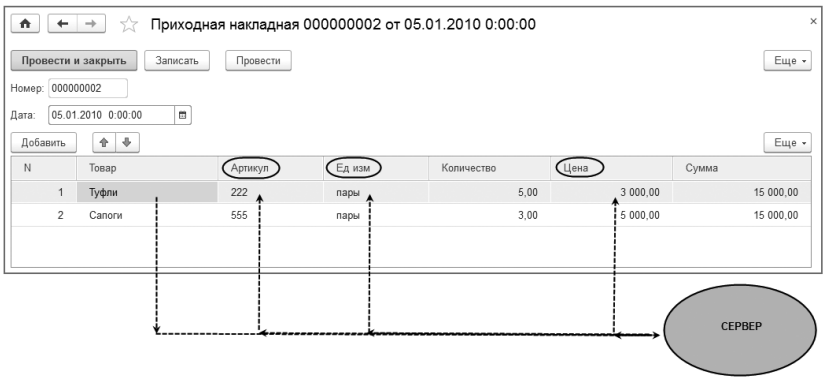


Рис. 4.19. Заполнение колонок табличной части в приходной накладной

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет втрое выше, так как при выборе товара в одной строке табличной части документа будет произведено *только одно обращение к серверу* (рис. 4.20).

Рис. 4.20. Показатели производительности

Таким образом, мы видим реальный выигрыш в производительности прикладного решения по сравнению с предыдущим вариантом.

Однако заметьте, что мы получали реквизиты товара по его ссылке. Эта возможность платформы очень удобная, но приводящая к загрузке всего объекта в память, что не всегда нужно. Допустим, нам нужно получить только три реквизита товара, а всего их – пятьдесят. Поэтому *эффективнее использовать запрос для получения реквизитов товара*.

Заметим, что здесь, так же как и в предыдущем варианте, при первом выборе товара может быть лишний серверный вызов за счет кеширования формой данных в списке выбора.

В связи с этим изменим функцию для получения реквизитов товара в модуле формы следующим образом (листинг 4.7).

Листинг 4.7. Функция «ПолучитьРеквизитыТовара()»

```
&НаСервереБезКонтекста
Функция ПолучитьРеквизитыТовара(ТоварСсылка)

	РеквизитыТовара = Новый Структура;
	Запрос = Новый Запрос;
	Запрос.Текст = "ВЫБРАТЬ
		|	Товары.Артикул,
		|	Товары.ЕдиницаИзмерения,
		|	Товары.Цена
		|ИЗ
		|	Справочник.Товары КАК Товары
		|ГДЕ
		|	Товары.Ссылка = &Ссылка";
	Запрос.УстановитьПараметр("Ссылка", ТоварСсылка);
	Выборка = Запрос.Выполнить().Выбрать();

	Если Выборка.Следующий() Тогда
		Артикул = Выборка.Артикул;
```

```
            ЕдиницаИзмерения = Выборка.ЕдиницаИзмерения;
            Цена = Выборка.Цена;

        Иначе
            Артикул = 0;
            ЕдиницаИзмерения = 0;
            Цена = 0;

        КонецЕсли;

        РеквизитыТовара.Вставить("Артикул", Артикул);
        РеквизитыТовара.Вставить("ЕдиницаИзмерения", ЕдиницаИзмерения);
        РеквизитыТовара.Вставить("Цена", Цена);

        Возврат РеквизитыТовара;

КонецФункции
```

В этой функции мы заполняем поля структуры реквизитов товара значениями соответствующих реквизитов, полученных при помощи запроса из справочника `Товары`. В запросе мы устанавливаем параметр `Ссылка`, равный переданной в функцию ссылке на элемент справочника (`ТоварСсылка`).

Резюме

В процессе работы прикладного решения любое обращение на сервер сказывается на его производительности.

Поэтому *нужно стараться получить всю необходимую информацию за один вызов сервера*.

Не нужно создавать универсальные процедуры для обхода клиент-серверной логики, например `ПолучитьРеквизитИзБазыДанных(Ссылка, ИмяРеквизита)`, и вызывать их всякий раз, когда понадобятся какие-либо реквизиты объекта.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй вариант за счет *меньшего количества серверных вызовов*.

Заметим, что объединять в один серверный вызов нужно логически связанные действия, иначе может возникнуть желание вообще все сделать за один вызов сервера, даже то, что логически между собой не связано.

Использование внеконтекстных серверных процедур в модуле формы

Серверные процедуры и функции могут исполняться с передачей контекста формы на сервер (директива компиляции `&НаСервере`) и без (директива компиляции `&НаСервереБезКонтекста`).

При контекстных серверных вызовах процедур в модуле формы форма передает свои данные (реквизиты) и элементы на сервер и обратно. Данные формы становятся доступны на сервере, но нужно понимать, что передача контекста формы на сервер – это целый непростой процесс.

Ведь контекст формы сначала на клиенте нужно подготовить к передаче, упаковать, а потом на сервере создать специальную среду, в которой будет исполняться код, развернуть весь полученный контекст, инициализировать им эту среду и т. д. После выполнения процедуры измененные данные снова «поедут» на клиент, если их нужно отображать в форме. На все это, естественно, тратится время и ресурсы системы.

Однако использование контекстных серверных процедур необходимо, когда нужно, например, поменять данные во всей табличной части или когда нужно при этом поработать с тем объектом, который отображается в форме, выполнить его экспортируемый метод и т. п.

Во внеконтекстных серверных процедурах недоступен контекст формы. Это значит, что при вызове этих процедур не происходит передача данных формы на сервер и обратно. Соответственно, производительность прикладного решения при использовании внеконтекстных серверных процедур будет выше.

Так что же использовать в том или ином случае: внеконтекстные или контекстные серверные процедуры? Решение зависит от конкретной задачи. Рассмотрим пример.

Предположим, в периодическом регистре сведений `Цены` содержатся розничные цены товаров из справочника `Товары`. Для расходования товаров существует документ `РасходнаяНакладная`, в табличную часть которого подбираются товары.

Хочется, чтобы пользователь, выбрав товар, видел в документе не только ссылку, но сразу же и цену выбранного товара из регистра сведений, актуальную на дату документа (рис. 4.21).

Рис. 4.21. Подстановка актуальной цены товара в расходную накладную

Таким образом, в тот момент, когда пользователь выбирает товар в колонке Товар, ссылающейся на справочник Товары, колонку Цена табличной части документа РасходнаяНакладная нужно заполнить актуальной ценой товара из регистра сведений.

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

Чтобы обеспечить подстановку актуальной цены в расходную накладную при выборе товара, мы будем «уходить» на сервер, там определять цену товара, подставлять ее в документ и затем возвращаться обратно на клиент. Ведь получить доступ к базе данных мы можем только на сервере – поэтому сразу же сделаем все, что нам нужно, прямо на сервере.

Для этого создадим форму документа и обработчик события ПриИзменении() для поля формы Товар, содержащегося в таблице формы Товары (листинг 4.8).

Листинг 4.8. Процедура «ТоварыТоварПриИзменении()»

```
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

    РозничнаяЦена();

КонецПроцедуры
```

В этом обработчике мы вызываем контекстную серверную процедуру РозничнаяЦена(). Эту процедуру мы поместим в модуле формы документа (листинг 4.9).

Листинг 4.9. Процедура «РозничнаяЦена()»

```
&НаСервере
Процедура РозничнаяЦена()

	Идентификатор = Элементы.Товары.ТекущаяСтрока;
	ДанныеТекущейСтроки = Объект.Товары.НайтиПоИдентификатору(Идентификатор);

	Отбор = Новый Структура;
	Отбор.Вставить("Товар", ДанныеТекущейСтроки.Товар);
	ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее(Объект.Дата, Отбор);
	ДанныеТекущейСтроки.Цена = ЗначенияРесурсов.Цена;

КонецПроцедуры
```

В этой процедуре мы получаем Идентификатор текущей строки таблицы формы. По этому идентификатору находим эту строку в объекте ДанныеФормыКоллекция, содержащем табличную часть документа – Объект.Товары. Таким образом мы получаем доступ к данным текущей строки таблицы формы в переменной ДанныеТекущейСтроки и через точку от нее можем обращаться к значению колонок таблицы.

Затем мы создаем структуру Отбор, содержащую отбор по измерению регистра Товар, и устанавливаем его равным ссылке на выбранный товар (ДанныеТекущейСтроки.Товар).

Затем мы выполняем метод менеджера регистра сведений Цены ПолучитьПоследнее() и присваиваем значение ресурса Цена наиболее поздней записи регистра на дату документа (Объект.Дата) для выбранного товара (Отбор) соответствующей колонке таблицы формы.

Заметьте, что значения реквизитов документа (Объект.Дата, Объект.Товары) и свойства элементов формы (Элементы.Товары.ТекущаяСтрока) доступны на сервере, так как при использовании директивы компиляции &НаСервере в процедуру передается весь контекст формы.

Этот пример можно посмотреть в демонстрационной конфигурации «02 (вар. 1) Использование внеконтекстных серверных процедур в модуле формы».

Запустим «1С:Предприятие», откроем расходную накладную, добавим запись в табличную часть документа и сделаем выбор из справочника товаров в колонке Товар. После этого колонка табличной части Цена автоматически заполнится последней ценой из регистра сведений, актуальной на дату документа для выбранного товара (рис. 4.22).

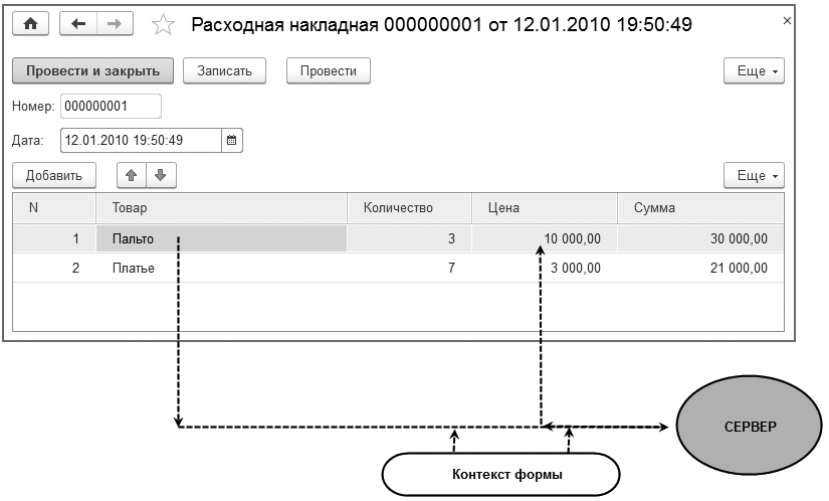


Рис. 4.22. Подстановка актуальной цены товара в расходную накладную

В результате использования процедуры РозничнаяЦена() для подстановки цены в документ будет происходить *одно обращение на сервер* (рис. 4.23).

Рис. 4.23. Показатели производительности

Заметим, что при первом выборе товара может быть лишний серверный вызов за счет автоматического кеширования формой данных в списке выбора. Но мы не будем исследовать эту ситуацию, так как этот вызов делается платформой, а нас интересуют вызовы, инициированные разработчиком.

Второй вариант решения

Один вызов сервера, происходящий при подстановке цены товара, неизбежен и оправдан. Но обратите внимание на объем данных, передаваемых на сервер и обратно, указанный на рисунке 4.21 (чтобы его увидеть, нужно включить соответствующие опции в окне настройки показателей производительности). Можно ли его уменьшить? Конечно! Действительно, зачем «гонять» весь контекст формы туда и обратно, когда нам нужна всего лишь одна цена товара?

На самом деле в функции для получения актуальной цены `РозничнаяЦена()` кроме ссылки на выбранный товар используется только один реквизит документа РасходнаяНакладная – Дата. Вместо того чтобы передавать весь контекст формы на сервер, можно передать эти значения в функцию в качестве параметров. Описание функции в модуле формы мы предварим директивой компиляции `&НаСервереБезКонтекста`. Таким образом мы не будем передавать данные формы на сервер, и наше прикладное решение будет работать быстрее.

И если раньше мы подставляли цену в документ на сервере, то теперь мы будем делать это на клиенте. Именно для этого действия сервер совершенно не нужен, это можно прекрасно сделать прямо на клиенте.

Для этого изменим обработчик события `ПриИзменении()` поля Товар табличной части документа РасходнаяНакладная следующим образом (листинг 4.10).

Листинг 4.10. Процедура «ТоварыТоварПриИзменении()»

```
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

	ДанныеТекущейСтроки = Элементы.Товары.ТекущиеДанные;
	ДанныеТекущейСтроки.Цена = РозничнаяЦена(ДанныеТекущейСтроки.Товар, Объект.Дата);

КонецПроцедуры
```

В этом обработчике мы получаем доступ к данным текущей строки таблицы формы в переменной `ДанныеТекущейСтроки` и через точку от нее можем обращаться к значению колонок таблицы. Ссылку на выбранный товар (`ДанныеТекущейСтроки.Товар`) и дату документа (`Объект.Дата`) мы передаем в функцию для получения актуальной

цены РозничнаяЦена() и присваиваем возвращенное значение колонке табличной части Цена.

В модуле формы документа поместим функцию РозничнаяЦена(), выполняющуюся на сервере без контекста формы (листинг 4.11).

Листинг 4.11. Функция «РозничнаяЦена()»

```
&НаСервереБезКонтекста
Функция РозничнаяЦена(ВыбранныйТовар, АктуальнаяДата)

    Отбор = Новый Структура;
    Отбор.Вставить("Товар", ВыбранныйТовар);
    ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее(АктуальнаяДата, Отбор);

    Возврат ЗначенияРесурсов.Цена;

КонецФункции
```

В этой функции мы создаем структуру Отбор, содержащую отбор по измерению регистра Товар, и устанавливаем отбор равным ссылке на выбранный товар. Затем при помощи метода ПолучитьПоследнее() мы возвращаем актуальную цену выбранного товара.

Запустим «1С:Предприятие», откроем расходную накладную и сделаем выбор из справочника товаров в колонке Товар табличной части документа. После этого колонка табличной части Цена автоматически заполнится последней ценой из регистра сведений, актуальной на дату документа для выбранного товара (рис. 4.24).

Этот пример можно посмотреть в демонстрационной конфигурации «02 (вар. 2) Использование внеконтекстных серверных процедур в модуле формы».

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но объем передаваемых на сервер данных будет значительно меньше (вместо 2481 – 1886 байт), так как при вызове функции РозничнаяЦена() мы использовали *внеконтекстный серверный вызов* и не передавали все данные формы на сервер (рис. 4.25).

Заметим, что здесь, так же как и в предыдущем варианте, при первом выборе товара может быть лишний серверный вызов за счет автоматического кеширования формой данных в списке выбора.

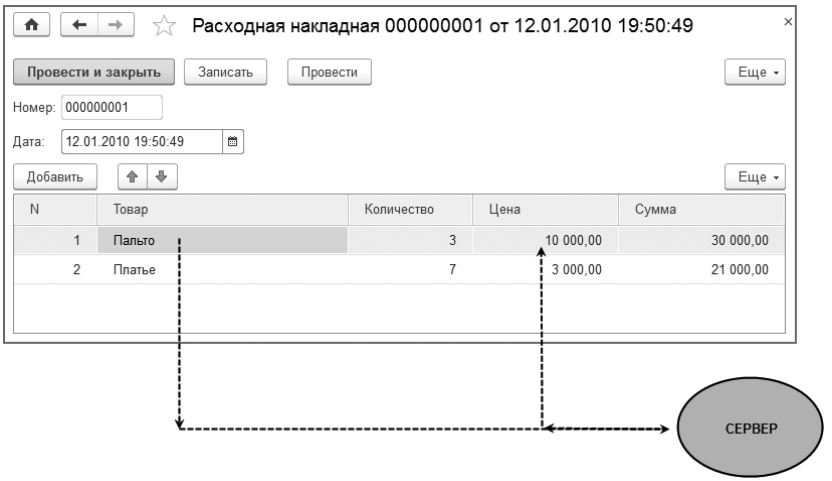


Рис. 4.24. Подстановка актуальной цены товара в расходную накладную

Рис. 4.25. Показатели производительности

Но, с другой стороны, использование процедур с параметрами тоже может иметь свои недостатки. Слишком большое количество параметров в процедуре может снижать производительность. «Слишком большое» – понятие относительное, конкретные числовые оценки дать нельзя. Но, например, если есть выбор: передать 100 параметров или передать всего 2 параметра, – то, конечно же, нужно передавать 2 параметра, а не 100. Поэтому разработчику всегда нужно искать компромисс между здравым смыслом («читабельностью» программы) и оптимизацией.

В связи с этим можно на клиенте формировать структуру параметров, передаваемых в функцию, и затем передавать ее целиком, одним параметром.

В данном конкретном случае мы передаем всего два параметра. Это нормально –можно передавать их по отдельности, и нет необходимости сокращать их количество. Но на этом примере мы покажем, как в принципе можно это сделать. Например, сгруппировать

логически связанные между собой параметры в один, который является структурой.

Итак, объединим два параметра, передаваемых в функцию для получения актуальной цены, в структуру, и передадим ее целиком как структуру параметров (листинги 4.12, 4.13).

Листинг 4.12 Процедура «ТоварыТоварПриИзменении()»

```
&НаКлиенте
Процедура ТоварыТоварПриИзменении(Элемент)

	ДанныеТекущейСтроки = Элементы.Товары.ТекущиеДанные;
	Отбор = Новый Структура("Товар", ДанныеТекущейСтроки.Товар);
	ПараметрыФункции = Новый Структура("Отбор, АктуальнаяДата", Отбор, Объект.Дата);
	ДанныеТекущейСтроки.Цена = РозничнаяЦена(ПараметрыФункции);

КонецПроцедуры
```

Листинг 4.13. Функция «РозничнаяЦена()»

```
&НаСервереБезКонтекста
Функция РозничнаяЦена(ПараметрыФункции)

	ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее(
		ПараметрыФункции.АктуальнаяДата, ПараметрыФункции.Отбор);

	Возврат ЗначенияРесурсов.Цена;

КонецФункции
```

Обратите внимание, если мы сравним объем передаваемых данных, то оказывается, что он больше, чем когда мы передавали параметры в функцию по отдельности (рис. 4.26).

Рис. 4.26. Показатели производительности

Получается, что для данного примера это действительно ненужная «оптимизация». Как мы и говорили, два параметра в этом примере – это хорошо, и мы даже получили увеличение объема передаваемых данных, когда упаковали их в одну структуру. Но когда параметров слишком много, ситуация может быть обратной. Поэтому в каждой ситуации нужно думать, как лучше поступить, и упаковка параметров в один должна быть логически понятна и объяснима.

Резюме

Применение внеконтекстных процедур позволяет значительно уменьшить объем передаваемых данных (трафик) между клиентом и сервером и уменьшить нагрузку на систему.

Контекстную передачу управления на сервер целесообразно использовать при работе с большими объемами данных (реквизитами формы типа табличных документов или коллекциями элементов типа `ДанныеФормыКоллекция`, `ДанныеФормыСтруктураСКоллекцией`, `ДанныеФормыДерево`). В этих случаях платформа «1С:Предприятие» самостоятельно оптимизирует объем передаваемых между клиентом и сервером данных (в обоих направлениях). При этом затраты ресурсов сервера на инициализацию контекста формы оправдываются существенным снижением трафика между клиентом и сервером и снижением числа вызовов сервера.

В остальных случаях нужно использовать внеконтекстные серверные процедуры, а если требуется передавать туда какие-то данные формы, то можно одно-два значения передать в качестве параметров.

Не нужно без необходимости использовать контекстные серверные процедуры, если в них не требуется использовать и изменять большинство реквизитов формы.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет использования *внеконтекстного серверного вызова*.

Использование клиентских процедур для небольших расчетов данных формы

В процессе разработки конфигурации часто возникает вопрос: «Где выполнять пересчет данных формы: на клиенте или на сервере?» Однозначного ответа тут нет.

С одной стороны, *на клиенте должно быть минимум кода,* потому что такова идеология клиентского приложения. Клиентского кода по определению не должно быть много. Основной подход заключается в том, что на клиенте должен исполняться только тот код, который

управляет формой и отображением данных в ней. Все расчеты должны выполняться на сервере.

С другой стороны, *нужно избегать лишних обращений на сервер.* И хотя процесс вычислений произойдет на сервере значительно быстрее, но сам вызов сервера, да еще с контекстом формы, снизит производительность прикладного решения.

Решение зависит от конкретной задачи. Рассмотрим пример.

Предположим, в форме документа об оказании услуг содержатся цена услуги, процент вознаграждения и сумма услуги. При изменении цены услуги нужно рассчитать сумму услуги, равную цене услуги плюс процент вознаграждения. Процент вознаграждения устанавливается в зависимости от цены услуги (рис. 4.27).

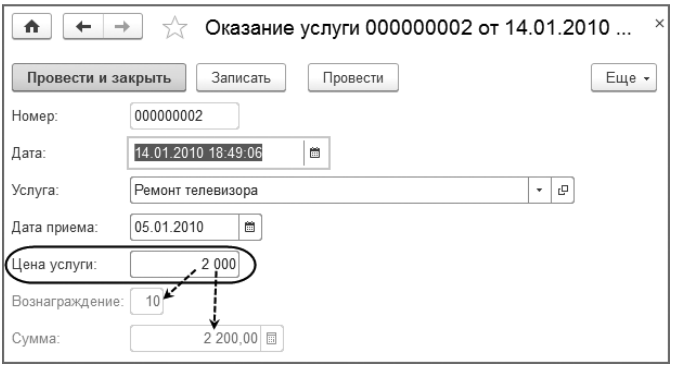


Рис. 4.27. Пересчет данных в документе при изменении цены услуги

Таким образом, в тот момент, когда пользователь изменит поле ЦенаУслуги документа ОказаниеУслуги, поля Вознаграждение и Сумма нужно рассчитать по описанному выше алгоритму.

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

Чтобы обеспечить перерасчет суммы услуги при изменении цены, мы сразу пойдем на сервер (ведь расчеты нужно делать на сервере), там рассчитаем вознаграждение и сумму услуги и поместим их в форму. Затем вернемся на клиент с готовой формой.

Для этого создадим форму документа ОказаниеУслуги и обработчик события ПриИзменении() для поля формы ЦенаУслуги (листинг 4.14).

Листинг 4.14. Процедура «ЦенаУслугиПриИзменении()»

```
&НаКлиенте
Процедура ЦенаУслугиПриИзменении(Элемент)

	РасчетСуммыУслуги();

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру РасчетСуммыУслуги(), исполняющуюся на сервере (листинг 4.15).

Листинг 4.15. Процедура «РасчетСуммыУслуги()»

```
&НаСервере
Процедура РасчетСуммыУслуги()

	Если Объект.ЦенаУслуги > 1000 Тогда
		Объект.Вознаграждение = 10;

	Иначе
		Объект.Вознаграждение = 5;

	КонецЕсли;

	Объект.Сумма = Объект.ЦенаУслуги + Объект.ЦенаУслуги * Объект.Вознаграждение / 100;

КонецПроцедуры
```

В процедуру расчета передается контекст формы, так как в ней изменяются реквизиты объекта Вознаграждение и Сумма. Отключим для соответствующих полей формы свойство Доступность, так как они рассчитываются при изменении поля ЦенаУслуги.

Этот пример можно посмотреть в демонстрационной конфигурации «03 (вар. 1) Использование клиентских процедур для небольших расчетов данных формы».

Запустим «1С:Предприятие», откроем документ об оказании услуг, выберем услугу и внесем цену услуги. После этого поля Вознаграждение и Сумма автоматически пересчитаются по заданному нами алгоритму (рис. 4.28).

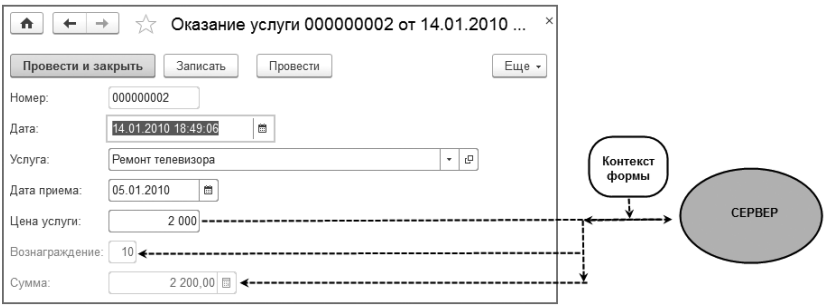


Рис. 4.28. Пересчет данных в документе при изменении цены услуги

В результате будет сделано *одно обращение на сервер*, кроме того, этот *вызов будет контекстным* (рис. 4.29).

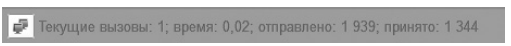


Рис. 4.29. Показатели производительности

Второй вариант решения

Чтобы оценить правильность своих действий, можно проанализировать, что же было сделано. Ход рассуждений может быть примерно таким:

- «Контекстный вызов. А он нужен? Можно было обойтись внеконтекстным вызовом?»
- «Внеконтекстный вызов. А он нужен? Вообще тут нужен вызов сервера? Есть что-то, что нельзя сделать на клиенте?»
- «Если это делать на клиенте, это сложные вычисления? Сложнее, чем если бы их делали на сервере?».

На самом деле обращения на сервер для пересчета данных формы могло бы и не быть. Как мы увидим ниже, более эффективно этот расчет будет работать на клиенте.

Поскольку все данные для пересчета доступны на клиенте (это поля основного реквизита формы Объект – ЦенаУслуги, Вознаграждение и Сумма), то нет смысла лишний раз обращаться на сервер. Код пересчета – небольшой, и он быстро выполнится из клиентской процедуры при изменении цены услуги.

Для этого изменим обработчик события ПриИзменении( ) поля ЦенаУслуги документа ОказаниеУслуги следующим образом (листинг 4.16).

Листинг 4.16. Процедура «ЦенаУслугиПриИзменении()»

```
&НаКлиенте
Процедура ЦенаУслугиПриИзменении(Элемент)

	Если Объект.ЦенаУслуги > 1000 Тогда
		Объект.Вознаграждение = 10;

	Иначе
		Объект.Вознаграждение = 5;

	КонецЕсли;

	Объект.Сумма = Объект.ЦенаУслуги + Объект.ЦенаУслуги * Объект.Вознаграждение / 100;

КонецПроцедуры
```

Запустим «1С:Предприятие», откроем документ об оказании услуг, выберем услугу и внесем цену услуги. После этого поля Вознаграждение и Сумма автоматически пересчитаются по заданному нами алгоритму.

Этот пример можно посмотреть в демонстрационной конфигурации «03 (вар. 2) Использование клиентских процедур для небольших расчетов данных формы».

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, так как мы выполнили перерасчет данных документа *на клиенте, без обращения к серверу.*

Резюме

Таким образом, если в процедуре пересчета требуется массированная обработка данных формы или если в ней используются свойства и методы объектов, недоступных на клиенте, то выполнять пересчет данных формы нужно на сервере. Для этого необходимо

поместить код в контекстную серверную процедуру модуля формы. Или поместить код в процедуру модуля объекта и вызывать этот метод объекта, предварительно конвертировав данные формы в объект.

Если в расчете участвует немного данных формы и все эти данные уже есть на клиенте, то лучше это делать в клиентской процедуре модуля формы.

Не следует выполнять на клиенте сложные алгоритмы, требующие значительных ресурсов компьютера. В таких случаях выполнение алгоритма на клиенте может занимать гораздо больше времени, чем передача управления с клиента на сервер, выполнение алгоритма на сервере и возврат результата обратно на клиент.

Клиентский код должен максимально быстро сделать что-то на клиенте, без обращения к серверу. Он не должен содержать большой объем бизнес-логики. Например, быстро пересчитать сумму по количеству и цене или рассчитать значение какого-то реквизита на основе другого и т. п.

Поэтому *не нужно без необходимости использовать контекстные серверные процедуры для небольших расчетов данных формы.*

Таким образом, из показанных вариантов решения задачи более эффективным будет второй вариант за счет пересчета данных формы *на клиенте, без обращения к серверу.*

Использование контекстных серверных процедур для пересчета данных коллекций форм

Теперь рассмотрим пример, когда пересчет данных формы более эффективно выполнять на сервере. Мы уже говорили в предыдущем примере, что на сервере вычисления необходимо выполнять, когда на клиенте нет нужных типов данных, или когда используется сложный алгоритм расчета, содержащий много кода, или когда требуется обратиться к объекту, содержащемуся в форме, напрямую.

Однако в данном примере будет рассмотрен другой вопрос: где наиболее эффективно выполнять пересчет большого количества строк табличной части – на сервере или на клиенте?

С одной стороны, на серверный контекстный вызов платформа тратит время и ресурсы. Желательно бы этого избегать.

С другой стороны, на клиенте присутствуют не все строки коллекций форм (табличных частей, таблиц значений), а только их видимая часть. Остальные «дочитываются» с сервера динамически, по мере того как пользователь прокручивает таблицу. Или по мере того как разработчик перебирает коллекцию данных в реквизите формы из встроенного языка.

Таким образом, при массовом пересчете на клиенте все строки табличной части будут получены с сервера на клиент и все вернутся обратно при следующем контекстном вызове сервера.

Посмотрим, какое решение более оптимально. Рассмотрим пример.

Предположим, в табличной части расходной накладной содержатся товары и их цены. При нажатии кнопки Пересчет цен, расположенной в форме документа, должно производиться повышение цен товаров на 10 %, а также пересчет сумм продажи товаров как произведения их количества и цены (рис. 4.30).

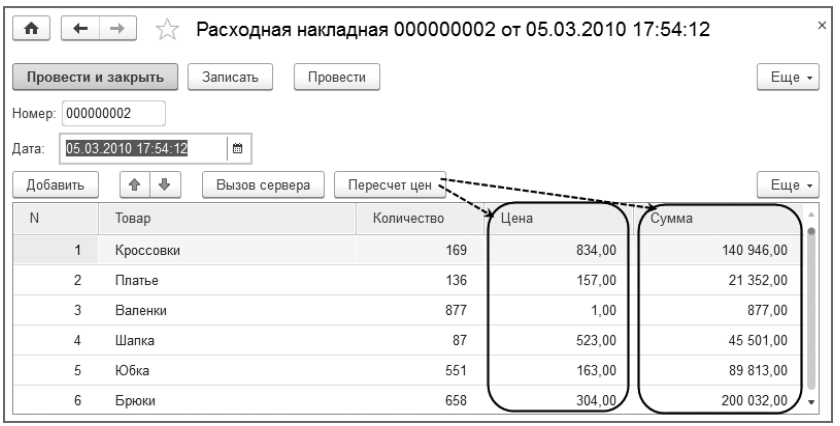


Рис. 4.30. Пересчет цен товаров в расходной накладной

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

Чтобы обеспечить пересчет цен товаров в табличной части расходной накладной, не будем вызывать сервер. Пересчитаем все цены на клиенте, ведь все необходимые данные на нем есть и обращаться к серверу вроде бы незачем.

Для этого создадим форму документа и ее команду ПересчетЦен. Перетащим команду в командную панель таблицы формы Товары. Обработчик команды заполним следующим образом (листинг 4.17).

Листинг 4.17. Обработчик команды «ПересчетЦен»

```
&НаКлиенте
Процедура ПересчетЦен(Команда)

    Для Каждого ТекСтрокаТовары Из Объект.Товары Цикл
        ТекСтрокаТовары.Цена = ТекСтрокаТовары.Цена * 1.1;
        ТекСтрокаТовары.Сумма = ТекСтрокаТовары.Количество * ТекСтрокаТовары.Цена;

    КонецЦикла;

КонецПроцедуры
```

В этом обработчике мы в цикле обходим табличную часть документа и пересчитываем цены и суммы товаров.

Запустим «1С:Предприятие», откроем документ РасходнаяНакладная № 2 (он содержит 1000 позиций товаров) и нажмем кнопку Пересчет цен. После этого цены и суммы товаров пересчитаются по заданному нами алгоритму (рис. 4.30).

Этот пример можно посмотреть в демонстрационной конфигурации «04 (вар. 1) Использование контекстных серверных процедур для пересчета данных коллекций форм».

Пересчет всех строк табличной части произойдет *непосредственно на клиенте*. Но при нажатии кнопки Пересчет цен произойдет *двадцать восемь вызовов сервера* (рис. 4.31).

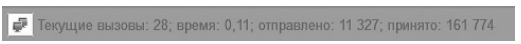


Рис. 4.31. Показатели производительности

Так происходит потому, что при открытии формы на клиент передается 35 строк табличной части, и затем, по мере программного обращения, остальные строки «дочитываются» с сервера порциями по 35 строк (1 015 = 28 * 35 + 35).

В процессе пересчета все недостающие (965 = 1 000 - 35) строки «приедут» с сервера на клиент. При следующем контекстном вызове все измененные данные, то есть все 1000 строк, «поедут» обратно на сервер (рис. 4.32).

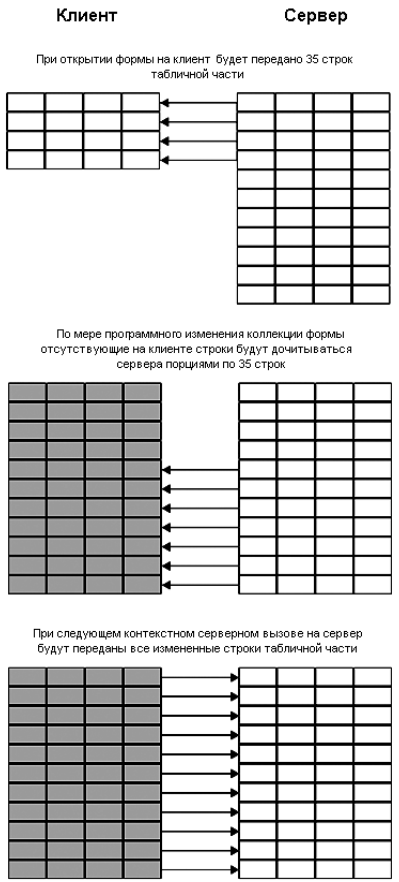


Рис. 4.32. Схема программного взаимодействия сервера и клиента

Например, сделаем команду Вызов сервера, после пересчета цен вызовем из нее пустую серверную контекстную процедуру и посмотрим на показатели производительности (рис. 4.33).

Рис. 4.33. Показатели производительности

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (рис. 4.34).

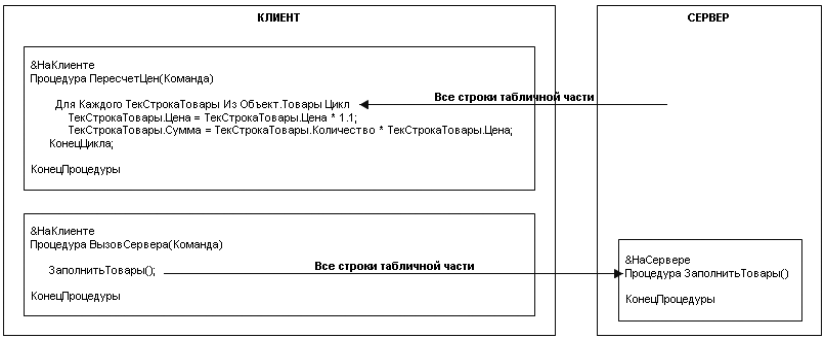


Рис. 4.34. Схема передачи данных между клиентом и сервером

Рассмотрим подробнее, сколько вызовов сервера при этом произойдет и каков будет объем передаваемых данных между клиентом и сервером. Мы специально рассматриваем пример документа с большой табличной частью, так как на нем лучше сравнивать показатели производительности.

Таблица 4.1. Показатели производительности

	Текущие вызовы	Отправлено	Принято
Открытие формы	2	1 811	16 343
Пересчет табличной части	28	11 327	161 774
Контекстный серверный вызов	1	205 591	1 106

При открытии формы 2 вызова сервера происходят при открытии формы в первый раз, при следующем открытии – 1 вызов.

Объем принятых данных отражает ситуацию, когда на клиент «приезжают» форма и «видимые» 35 строк табличной части.

При пересчете табличной части 28 вызовов сервера тратятся на «дочитывание» строк табличной части документа на клиент порциями по 35 строк.

Объем принятых данных отражает ситуацию, когда табличная часть «доехала» в форму документа на клиент.

Объем отправленных данных отражает ситуацию, когда форма сообщает серверу, какие строки и в каком порядке получить.

При следующем контекстном вызове объем отправленных данных отражает ситуацию, когда все 1000 измененных строк табличной части «уезжают» на сервер.

Таким образом, объем принятой информации – 16 343 + 161 774 = 178 117 байтов, а объем отправленной – 205 591 байт.

Если сравнить объем принятых и переданных данных, то получится, что форма и 1000 строк, переданные на клиент, меньше, чем 1000 строк, переданных с клиента на сервер. Это действительно так, потому что обратно передаются не только строки, но и служебная информация о том, что строка изменена.

Таким образом, в первом случае 161 774 байта информации передается на клиент и 205 591 байт передается на сервер. Как видите, объем передаваемых данных будет довольно большим (161 774 + 205 591 = 367 365 байтов). На самом деле, как будет показано ниже, более эффективно этот расчет будет работать на сервере.

Второй вариант решения

Форма всегда создается на сервере, а не на клиенте. Таким образом, при открытии формы все данные для пересчета уже есть на сервере. Поэтому лучше выполнять пересчет строк табличной части на сервере, а не на клиенте, так как строки для пересчета не будут передаваться на клиент.

Поэтому в данном решении мы сразу «уйдем» на сервер вместе с контекстом формы и там все пересчитаем. И затем вернемся на клиент.

Для этого изменим обработчик команды ПересчетЦен документа РасходнаяНакладная следующим образом (листинг 4.18).

Листинг 4.18. Обработчик команды «ПересчетЦен»

```
&НаКлиенте
Процедура ПересчетЦен(Команда)

    ПересчетЦенНаСервере();

КонецПроцедуры
```

В этом обработчике мы вызываем серверную контекстную процедуру ПересчетЦенНаСервере(). Поместим ее в модуле формы (листинг 4.19).

Листинг 4.19. Процедура «ПересчетЦенНаСервере()»

```
&НаСервере
Процедура ПересчетЦенНаСервере()

    Для Каждого ТекСтрокаТовары Из Объект.Товары Цикл
        ТекСтрокаТовары.Цена = ТекСтрокаТовары.Цена * 1.1;
        ТекСтрокаТовары.Сумма = ТекСтрокаТовары.Количество * ТекСтрокаТовары.Цена;

    КонецЦикла;

КонецПроцедуры
```

Запустим «1С:Предприятие», откроем документ РасходнаяНакладная № 2 (он содержит 1000 позиций товаров) и нажмем кнопку Пересчет цен. После этого цены и суммы товаров пересчитаются по заданному нами алгоритму.

Посмотрим на схему программного взаимодействия клиента и сервера в этом случае (рис. 4.35).

Этот пример можно посмотреть в демонстрационной конфигурации «04 (вар. 2) Использование контекстных серверных процедур для пересчета данных коллекций форм».

Функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, хотя мы и *выполнили контекстный серверный вызов.*

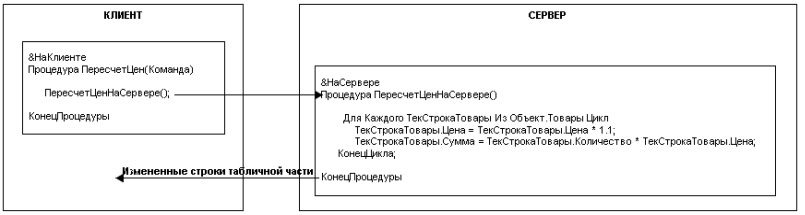


Рис. 4.35. Схема программного взаимодействия сервера и клиента

Посмотрим теперь на показатели производительности при нажатии кнопки Пересчет цен (рис. 4.36).

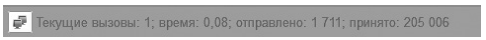


Рис. 4.36. Показатели производительности

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (первый вариант решения для сравнения приведен слева) – рис. 4.37.

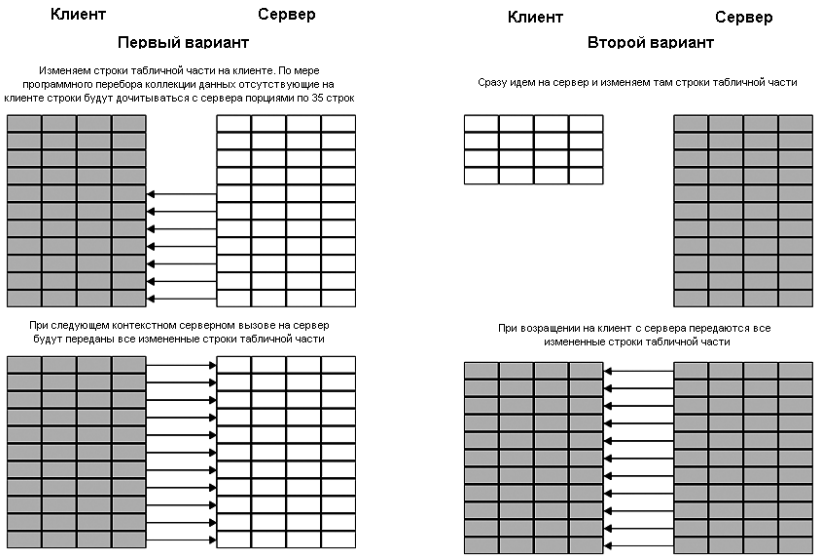


Рис. 4.37. Схема передачи данных между клиентом и сервером

Рассмотрим подробнее, сколько вызовов сервера при этом произойдет и каков будет объем передаваемых между клиентом и сервером данных. Мы специально рассматриваем пример документа с большой табличной частью, так как на нем лучше сравнивать показатели производительности.

Таблица 4.2. Показатели производительности

	Текущие вызовы	Отправлено	Принято
Пересчет табличной части на сервере и возврат на клиент	1	1 711	205 006

При пересчете табличной части на сервере объем принятых данных отражает ситуацию, когда все 1000 измененных строк табличной части «приезжают» на клиент.

Таким образом, во втором варианте передается 205 006 байтов информации против 367 365 байтов в первом. Заметим, что мы здесь сравниваем только объем передаваемых данных, инициируемый разработчиком.

Резюме

Все данные формы изначально доступны на сервере при ее открытии. В частности, на сервере присутствуют все строки табличных частей, а на клиенте – только их видимая часть.

Поэтому *изменять данные коллекций форм (табличные части, таблицы значений) лучше на сервере. Если же аналогичные действия выполнять на клиенте, то все данные коллекций форм будут переданы на клиент, хотя они не требуются для отображения в форме.*

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет того, что данные табличной части изменяются на сервере, а значит, не происходит ненужная предварительная передача всей табличной части на клиент.

Управление открываемой формой путем передачи параметров

Часто бывает нужно открыть форму не интерактивно, а из встроенного языка. Как правило, при этом в зависимости от различных условий форма должна иметь тот или иной вид.

Чтобы открыть форму, используется метод глобального контекста `ОткрытьФорму()`. При этом в форму передаются параметры – стандартные или созданные разработчиком. Они используются для того, чтобы открыть форму в некотором нужном состоянии.

Параметры формы представляют собой структуру, каждый элемент которой описывает один параметр формы. Эта структура передается в метод `ОткрытьФорму()` вторым параметром, и в результате форма открывается за один серверный вызов в нужном состоянии.

Другим способом является получение формы методом `ПолучитьФорму()`. Используя возвращенное этим методом значение объекта `УправляемаяФорма`, можно обращаться к ее свойствам и методам, а также к свойствам и методам ее элементов, чтобы подготовить форму к открытию в нужном состоянии. И затем открыть форму методом `Открыть()`.

Какой же способ наиболее эффективен? Рассмотрим пример.

Предположим, в конфигурации существуют иерархический справочник `Товары` и документ `ПриходнаяНакладная` с табличной частью `Товары`, содержащей перечень приходуемых товаров. При нажатии кнопки `Подбор`, расположенной в форме документа, должен производиться подбор товаров в табличную часть приходной накладной. При этом форма выбора из справочника `Товары` должна открываться в режиме множественного выбора, и иерархический список товаров должен быть представлен в виде дерева (рис. 4.38).

Заметим, что если форма выбора всегда должна открываться в заданном виде, то можно задать свойства формы в конфигураторе и ничего не программировать. Однако форма выбора может открываться из разных мест конфигурации, и ее внешний вид может зависеть от различных условий. В таком случае нужно программно формировать внешний вид формы при ее открытии.

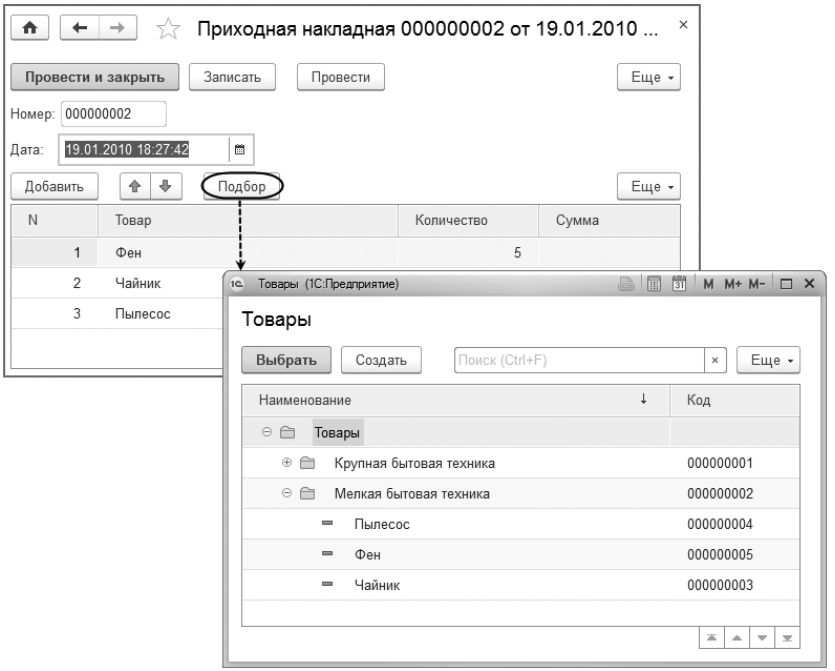


Рис. 4.38. Подбор товаров в табличную часть приходной накладной

Последний случай мы сейчас рассмотрим и изучим, как всегда, возможные варианты решения этой задачи.

Первый вариант решения

В данном решении мы получим форму на клиенте, установим нужные свойства этой формы и затем откроем ее.

Чтобы обеспечить подбор товаров в табличную часть приходной накладной, создадим форму документа и ее команду Подбор. Перетащим команду в командную панель таблицы формы Товары. Обработчик команды Подбор заполним следующим образом (листинг 4.20).

Листинг 4.20. Обработчик команды «Подбор»

```
&НаКлиенте
Процедура Подбор(Команда)

	ФормаВыбора = ПолучитьФорму("Справочник.Товары.ФормаВыбора", , Элементы.Товары);
	ФормаВыбора.Элементы.Список.МножественныйВыбор = Истина;
	ФормаВыбора.Элементы.Список.Отображение = ОтображениеТаблицы.Дерево;
	ФормаВыбора.Открыть();

КонецПроцедуры
```

В этом обработчике мы получаем основную форму выбора справочника Товары как подчиненную таблице Товары формы документа ПриходнаяНакладная. Таким образом, переменная ФормаВыбора будет содержать объект УправляемаяФорма. Используя коллекцию элементов (Элементы) этой формы, мы устанавливаем свойство МножественныйВыбор таблицы формы Список в значение Истина и свойство Отображение в значение системного перечисления ОтображениеТаблицы.Дерево.

Тем самым мы добьемся того, что форма, отражающая иерархический список товаров, будет открываться в режиме множественного выбора и таблица списка будет представлена в виде дерева. И затем мы открываем форму выбора методом Открыть().

При выборе из формы выбора справочника выбранное значение будет передано в обработчик события ОбработкаВыбора таблицы формы Товары приходной накладной, так как она является владельцем открытой формы выбора. Причем при множественном выборе форма возвращает не один элемент, а массив элементов.

Поэтому создадим обработчик события ОбработкаВыбора таблицы формы Товары и заполним его следующим образом (листинг 4.21).

Листинг 4.21. Обработчик события «ОбработкаВыбора» таблицы «Товары»

```
&НаКлиенте
Процедура ТоварыОбработкаВыбора(Элемент, ВыбранноеЗначение, СтандартнаяОбработка)

	Для Каждого ВыбранныйЭлемент Из ВыбранноеЗначение Цикл
		НоваяСтрока = Объект.Товары.Добавить();
		НоваяСтрока.Товар = ВыбранныйЭлемент;

	КонецЦикла;

КонецПроцедуры
```

В этом обработчике мы организуем цикл обхода массива переданных элементов (ВыбранноеЗначение). И в этом цикле добавим новую строку в табличную часть документа (Объект.Товары) и присвоим ее полю Товар значение очередного элемента массива выбранных товаров.

Запустим «1С:Предприятие», откроем приходную накладную и нажмем кнопку Подбор. После этого форма выбора из справочника товаров будет открыта в режиме множественного выбора и иерархический список товаров будет представлен в виде дерева. При нажатии кнопки Выбрать выделенные записи из справочника товаров будут добавлены в табличную часть приходной накладной (рис. 4.38).

Этот пример можно посмотреть в демонстрационной конфигурации «05 (вар. 1) Управление открываемой формой путем передачи параметров».

Если мы посмотрим на показатели производительности, то увидим, что при открытии формы выбора в процедуре Подбор() произойдут *два вызова сервера* (рис. 4.39).

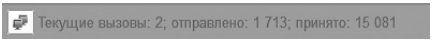


Рис. 4.39. Показатели производительности

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.40).

Один вызов происходит при выполнении метода ПолучитьФорму(), и он оправдан, так как его делает сама платформа.

А вот второго вызова, который происходит при изменении свойства Отображение таблицы формы Список, могло бы и не быть.

Ниже мы покажем, как этого избежать.

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

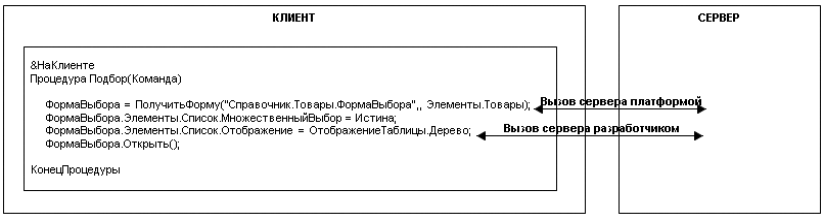


Рис. 4.40. Схема программного взаимодействия сервера и клиента

Второй вариант решения

На самом деле более эффективно открывать форму методом `ОткрытьФорму()`, при этом передавая в нее параметры. Параметры будут доступны в обработчике события формы выбора ПриСозданииНаСервере, где и нужно управлять состоянием формы при ее открытии.

Для этого изменим обработчик команды Подбор документа ПриходнаяНакладная следующим образом (листинг 4.22).

Листинг 4.22. Обработчик команды «Подбор»

```
&НаКлиенте
Процедура Подбор(Команда)

    ПараметрыФормы = Новый Структура;
    ПараметрыФормы.Вставить("МножественныйВыбор", Истина);
    ПараметрыФормы.Вставить("РежимОтображения", "Дерево");

    ОткрытьФорму("Справочник.Товары.ФормаВыбора", ПараметрыФормы, Элементы.Товары);

КонецПроцедуры
```

В этом обработчике мы создаем структуру ПараметрыФормы и добавляем в нее элементы МножественныйВыбор со значением Истина и РежимОтображения со значением Дерево.

Причем МножественныйВыбор – это стандартный параметр, поставляемый расширением формы для динамического списка, который при открытии формы выбора устанавливает соответствующее свойство для таблицы динамического списка. Это платформа делает автоматически.

А параметр РежимОтображения мы создаем сами, чтобы передать в обработчик события ПриСозданииНаСервере формы выбора справочника режим отображения таблицы динамического списка в виде дерева.

Затем мы открываем основную форму выбора справочника Товары как подчиненную таблице Товары формы документа ПриходнаяНакладная методом ОткрытьФорму(), который производит одно обращение к серверу.

Для анализа параметра РежимОтображения создадим форму выбора справочника Товары и обработчик события формы ПриСозданииНаСервере. Заполним его следующим образом (листинг 4.23).

Листинг 4.23. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Если Параметры.РежимОтображения = "Дерево" Тогда
        Элементы.Список.Отображение = ОтображениеТаблицы.Дерево;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы анализируем значение параметра РежимОтображения и в соответствии с ним устанавливаем свойство Отображение таблицы формы Список в значение системного перечисления ОтображениеТаблицы.Дерево.

Запустим «1С:Предприятие», откроем приходную накладную и нажмем кнопку Подбор. После этого форма выбора из справочника товаров будет открыта в режиме множественного выбора, и иерархический список товаров будет представлен в виде дерева. При нажатии кнопки Выбрать выделенные записи из справочника товаров будут добавлены в табличную часть приходной накладной.

Этот пример можно посмотреть в демонстрационной конфигурации «05 (вар. 2) Управление открываемой формой путем передачи параметров».

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, так как для открытия формы выбора при помощи метода `ОткрытьФорму()` потребовался *только один вызов сервера* (рис. 4.41).

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Рис. 4.41. Показатели производительности

Резюме

Чтобы открыть форму в некотором нужном состоянии, *рекомендуется использовать метод* `ОткрытьФорму()` *и при этом передавать в него параметры. Форма будет открыта за один серверный вызов.*

Не рекомендуется открывать форму с помощью метода `ПолучитьФорму()` *и затем обращаться к ее свойствам и методам, так как изменение свойств и методов формы на клиенте может привести к лишним обращениям на сервер.*

Кроме того, программный код будет выглядеть при этом менее стройным и читаемым. Он может вообще перестать работать, если, например, в форме переименовали какой-то реквизит, к которому обращались снаружи.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет открытия формы с передачей в нее параметров *за один серверный вызов.*

Реализация функциональности в клиентских и серверных обработчиках событий формы в зависимости от их назначения

Для реализации функциональности форм разработчик может использовать клиентские и серверные обработчики событий формы. Но делать это нужно правильно и по назначению.

Например, при интерактивном или программном создании формы с существующими в информационной базе данными вызывается

серверное событие ПриЧтенииНаСервере. Затем для всех форм вызывается серверное событие ПриСозданииНаСервере. В обработчиках этих двух событий можно и нужно подготавливать форму к работе.

Затем вызывается клиентское событие формы ПриОткрытии, в обработчике которого можно выполнять действия, доступные только на клиенте, например общение с пользователем.

Часто бывает нужно задать функциональность формы сразу при ее открытии – например, заполнить какие-то реквизиты формы или установить какие-то свойства ее элементов и т. п. В обработчике какого же события: ПриСозданииНаСервере, ПриЧтенииНаСервере или ПриОткрытии – это нужно делать? Рассмотрим пример.

Предположим, в периодическом регистре сведений Цены содержатся розничные цены товаров из справочника Товары. При открытии формы уже существующего товара в поле Розничная цена должна подставляться цена из регистра сведений, и это поле должно быть недоступно для редактирования (рис. 4.42).

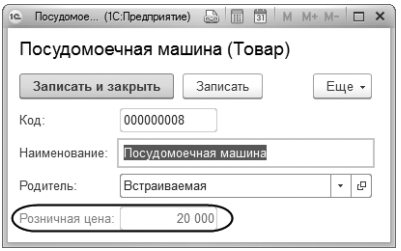


Рис. 4.42. Форма редактирования существующего товара

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

Чтобы обеспечить нужную функциональность формы при ее открытии, все действия будем выполнять в обработчике события ПриОткрытии. Подходящее название, зачем задумываться о назначении других обработчиков? «При открытии» – название события говорит само за себя, значит, в нем все и сделаем.

Для этого создадим форму элемента справочника Товары. Создадим реквизит формы РозничнаяЦена типа Число и перетащим его в дерево элементов формы. Затем создадим обработчик события формы ПриОткрытии и заполним его следующим образом (листинг 4.24).

Листинг 4.24. Обработчик события «ПриОткрытии»

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)

    Если НЕ Параметры.Ключ.Пустая() Тогда
        РозничнаяЦена = РозничнаяЦена(Объект.Ссылка);
        Элементы.РозничнаяЦена.Доступность = Ложь;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы анализируем значение параметра формы Ключ. Если ссылка, содержащаяся в нем, непустая, то, значит, товар уже существует. Мы узнаем его цену на текущую дату из регистра сведений и присваиваем ее реквизиту РозничнаяЦена. Также мы устанавливаем для поля формы РозничнаяЦена свойство Доступность в значение Ложь.

Теперь поместим в модуль формы функцию для получения актуальной цены товара из регистра сведений, выполняющуюся на сервере без контекста формы (листинг 4.25).

Листинг 4.25. Функция «РозничнаяЦена()»

```
&НаСервереБезКонтекста
Функция РозничнаяЦена(ТоварСсылка)

    Отбор = Новый Структура;
    Отбор.Вставить("Товар", ТоварСсылка);
    ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее( , Отбор);

    Возврат ЗначенияРесурсов.Цена;

КонецФункции
```

В этой функции мы создаем структуру Отбор, содержащую отбор по измерению регистра Товар, и устанавливаем его равным ссылке на товар, переданной в функцию. Затем при помощи метода ПолучитьПоследнее() мы возвращаем последнюю на текущую дату цену товара.

Этот пример можно посмотреть в демонстрационной конфигурации «06 (вар. 1) Реализация функциональности в клиентских и серверных обработчиках событий формы в зависимости от их назначения».

Запустим «1С:Предприятие» и откроем форму редактирования одного из товаров. В открывшейся форме поле Розничная цена автоматически заполнится последней ценой этого товара из регистра сведений, при этом поле Розничная цена будет недоступно для редактирования (рис. 4.42).

Однако при открытии формы редактирования существующего товара будут сделаны *два обращения на сервер* (рис. 4.43).

Рис. 4.43. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.44).

Один вызов происходит при открытии формы элемента справочника, и его делает сама платформа.

А вот второго вызова, который происходит при подстановке цены из регистра сведений для существующего товара, могло бы и не быть. Этот лишний серверный вызов произошел с клиента, из обработчика события ПриОткрытии.

Ниже мы покажем, как этого избежать.

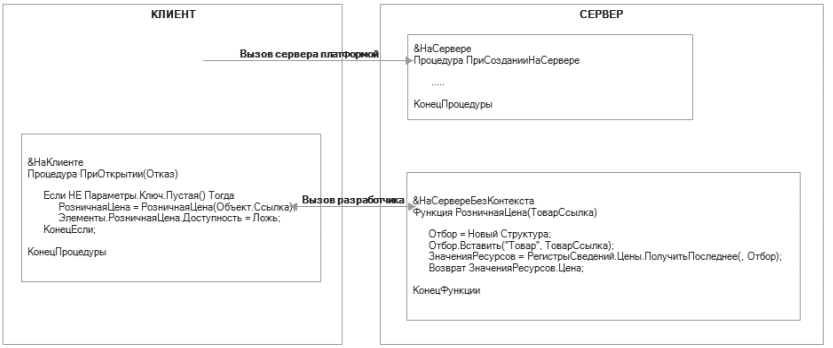


Рис. 4.44. Схема программного взаимодействия сервера и клиента

Второй вариант решения

На самом деле, когда открывается форма с существующими в информационной базе данными, вызывается серверное событие ПриЧтенииНаСервере. В обработчике этого события, в параметре ТекущийОбъект, нам доступен прикладной объект, содержащийся в форме, со всей его функциональностью. Таким образом, здесь мы можем подготовить данные формы, зависящие от данных объекта, к открытию.

Затем вызывается серверное событие формы ПриСозданииНаСервере. Это событие не зависит от того, какие данные форма отображает, и вызывается при открытии у всех форм. Именно в обработчике этого события нужно полностью подготавливать саму форму, ее внешнее представление, к открытию. Для подготовки данных формы лучше использовать событие ПриЧтенииНаСервере.

Таким образом, к моменту передачи формы на клиент и возникновению события ПриОткрытии вся подготовительная работа по открытию формы будет уже сделана. Останется только выполнить какое-то общение с пользователем (показать предупреждение, задать вопрос и т. п.), если это нужно.

В данном решении мы используем обработчик события ПриЧтенииНаСервере. Это событие вызывается только для существующих объектов, при создании новых оно не вызывается.

Поэтому для реализации нужной функциональности формы мы можем стандартно сконфигурировать поле РозничнаяЦена недоступным.

Итак, создадим обработчик события формы элемента справочника Товары ПриЧтенииНаСервере и переместим в него код из функции РозничнаяЦена() (листинг 4.26).

Листинг 4.26. Обработчик события «ПриЧтенииНаСервере»

```
&НаСервере
Процедура ПриЧтенииНаСервере(ТекущийОбъект)

    Отбор = Новый Структура;
    Отбор.Вставить("Товар", ТекущийОбъект.Ссылка);
    ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее( , Отбор);
    РозничнаяЦена = ЗначенияРесурсов.Цена;

КонецПроцедуры
```

Отключим свойство Доступность для поля РозничнаяЦена. Поскольку обработчик ПриЧтенииНаСервере будет отрабатывать только для существующих объектов, то для существующих товаров мы заполняем поле РозничнаяЦена последней ценой текущего товара из регистра сведений.

Этот пример можно посмотреть в демонстрационной конфигурации «06 (вар. 2) Реализация функциональности в клиентских и серверных обработчиках событий формы в зависимости от их назначения».

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, так как не будет лишних вызовов сервера с клиента. В результате та же функциональность формы будет реализована *за один серверный вызов* (рис. 4.45).

Рис. 4.45. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Резюме

Для реализации функциональности форм разработчик должен правильно и по назначению использовать клиентские и серверные обработчики событий формы.

Большая часть кода должна быть реализована в серверных обработчиках событий формы. Например, форма должна быть максимально подготовлена для открытия в обработчиках событий ПриЧтенииНаСервере *и* ПриСозданииНаСервере*, а в обработчике* ПриОткрытии *нужно выполнять только действия, недоступные на сервере: показать предупреждение, задать вопрос и т. п.*

При открытии формы настоятельно не рекомендуется выполнять обращения к серверу из кода модуля формы в обработчиках клиентских событий формы, таких как ПриОткрытии *и* ПриПовторномОткрытии. *При необходимости обращения из них к серверным данным следует размещать эти данные в реквизитах формы, в обработчике события* ПриСозданииНаСервере.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет реализации функциональности формы в обработчике события ПриЧтенииНаСервере.

Использование стандартных полей запроса в динамических списках на клиенте

Формы списка объектов конфигурации построены с помощью динамического списка. Динамический список формируется путем запроса к основной таблице, указанной в соответствующем свойстве реквизита типа ДинамическийСписок, или путем произвольного запроса к базе данных.

Обычно далеко не все поля динамического списка используются для отображения в форме. Например, в форме списка справочника это поля Код и Наименование плюс реквизиты, созданные разработчиком. В запросе на сервере из таблицы справочника могут быть выбраны и переданы на клиент и другие его стандартные реквизиты. Например, для иерархического справочника это поля Ссылка, ЭтоГруппа, Родитель, Предопределенный и т. п.

Особенность в том, что запросом выбираются и передаются на клиент значения только тех реквизитов, в свойствах которых стоит отметка Использовать всегда или они отображаются в форме (рис. 4.46).

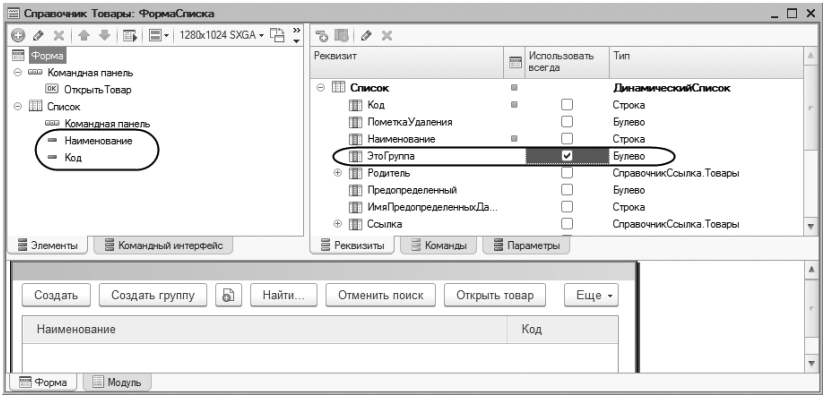


Рис. 4.46. Форма списка иерархического справочника

Таким образом, при помощи свойства полей динамического списка Использовать всегда можно получить значения реквизитов объекта на клиенте, не обращаясь лишний раз к серверу.

А можно «пойти» на сервер и получить эти реквизиты по ссылке на объект, используя свойство таблицы динамического списка `ТекущаяСтрока`.

Какой же способ наиболее эффективен? Рассмотрим этот вопрос на примере открытия текущего элемента из формы списка иерархического справочника.

Предположим, в конфигурации существует иерархический справочник Товары. При открытии формы элемента справочника из формы списка нужно проанализировать, является ли выбранный элемент группой, и в зависимости от этого открывать форму группы или форму элемента справочника.

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

В данном решении мы будем получать ссылку на текущий элемент, отображаемый в списке, затем «пойдем» с ней на сервер, чтобы из информационной базы узнать, является элемент группой или нет. В зависимости от этого, вернувшись на клиент, будем открывать либо форму элемента, либо форму группы.

Чтобы программно открывать форму элемента, создадим форму списка справочника Товары и ее команду ОткрытьТовар. Перетащим команду в командную панель формы. Обработчик команды ОткрытьТовар заполним следующим образом (листинг 4.27).

Листинг 4.27. Обработчик команды «Открыть товар»

```
&НаКлиенте
Процедура ОткрытьТовар(Команда)

    ССылкаНаЭлементСправочника = Элементы.Список.ТекущаяСтрока;
    ПараметрыФормы = Новый Структура("Ключ", ССылкаНаЭлементСправочника);

    Если ЕслиЭтоГруппа(ССылкаНаЭлементСправочника) Тогда
        ОткрытьФорму("Справочник.Товары.ФормаГруппы", ПараметрыФормы);

    Иначе
        ОткрытьФорму("Справочник.Товары.ФормаОбъекта", ПараметрыФормы);

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике, используя свойство ТекущаяСтрока таблицы формы Список, отражающей данные динамического списка, мы получаем ссылку на текущий элемент справочника. Значение ссылки мы присваиваем параметру формы Ключ. И в зависимости от того, является ли данный элемент группой, открываем форму группы или форму элемента справочника с этим параметром.

Для определения того, является ли элемент справочника группой по переданной ссылке, поместим в модуле формы функцию, исполняющуюся на сервере без контекста формы (листинг 4.28).

Листинг 4.28. Функция «ЕслиЭтоГруппа»

```
&НаСервереБезКонтекста
Функция ЕслиЭтоГруппа(Ссылка)

    Возврат Ссылка.ЭтоГруппа;

КонецФункции
```

Запустим «1С:Предприятие», выделим группу в форме списка товаров и нажмем кнопку Открыть товар. В результате будет открыта форма группы справочника товаров. Затем выделим элемент справочника и нажмем кнопку Открыть товар. В результате будет открыта форма элемента справочника товаров.

Этот пример можно посмотреть в демонстрационной конфигурации «07 (вар. 1) Использование стандартных полей запроса в динамических списках на клиенте».

Однако в обоих случаях при открытии формы будут сделаны *два обращения на сервер* (рис. 4.47).

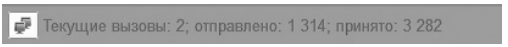


Рис. 4.47. Показатели производительности

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.48).

Второй вызов происходит при открытии формы элемента или группы справочника, и его делает сама платформа.

А вот первого вызова, который происходит при определении, является ли текущий элемент группой, могло бы и не быть. Ниже мы покажем, как этого избежать.

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

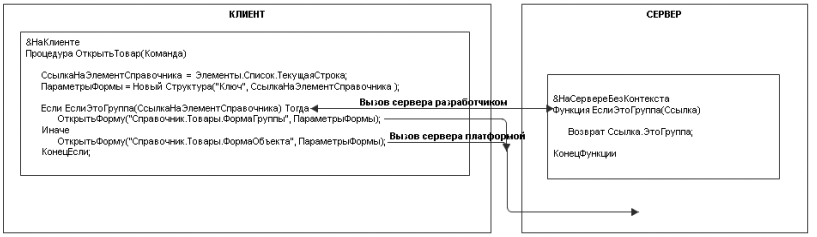


Рис. 4.48. Схема программного взаимодействия сервера и клиента

Второй вариант решения

В данном решении мы не будем специально «ходить» на сервер и определять, является текущий элемент группой или нет, потому что эти данные уже есть в динамическом списке. Просто сделаем так, чтобы они присутствовали в нем, даже несмотря на то что они не отображаются в форме. Тогда все, что нам нужно знать об этом элементе, мы сможем выяснить прямо на клиенте, без вызова сервера.

Установим флажок Использовать всегда для поля ЭтоГруппа в свойствах основного реквизита формы Список (рис. 4.46). Это значит, что данные этого поля будут передаваться на клиент, даже несмотря на то что в форме они нигде не отображаются. Такая возможность сделана как раз на случай только программного использования этих данных.

Теперь изменим обработчик команды ОткрытьТовар следующим образом (листинг 4.29).

Листинг 4.29. Обработчик команды «Открыть товар»

```
&НаКлиенте
Процедура ОткрытьТовар(Команда)

	ССылкаНаЭлементСправочника = Элементы.Список.ТекущаяСтрока;
	ПараметрыФормы = Новый Структура("Ключ", ССылкаНаЭлементСправочника);

	Если Элементы.Список.ТекущиеДанные.ЭтоГруппа Тогда
		ОткрытьФорму("Справочник.Товары.ФормаГруппы", ПараметрыФормы);

	Иначе
		ОткрытьФорму("Справочник.Товары.ФормаОбъекта", ПараметрыФормы);

	КонецЕсли;

КонецПроцедуры
```

В этом обработчике для определения того, является ли элемент справочника группой, мы используем свойство ТекущиеДанные таблицы формы Список, отражающей данные динамического списка. И затем через точку от него обращаемся к полю ЭтоГруппа текущей строки этой таблицы.

Этот пример можно посмотреть в демонстрационной конфигурации «07 (вар. 2) Использование стандартных полей запроса в динамических списках на клиенте».

Поскольку значение этого поля уже доступно на клиенте, нам не понадобится лишний раз обращаться на сервер. В результате та же функциональность будет реализована *за один серверный вызов*, и производительность прикладного решения будет выше (рис. 4.49).

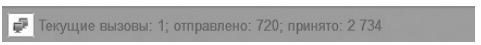


Рис. 4.49. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Резюме

В формах списка объектов конфигурации, построенных с помощью динамического списка, часто бывает нужно использовать значения стандартных реквизитов текущего объекта списка.

Для этого *нужно получать значения стандартных реквизитов объекта в динамических списках на клиенте, устанавливая флажок* Использовать всегда*, вместо того чтобы обращаться на сервер за этими значениями*.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет получения реквизитов динамического списка *на клиенте без обращения к серверу.*

Тут есть один интересный момент. Заметьте, что в этой ситуации можно было бы использовать метод ПоказатьЗначение(, Элементы.Список.ТекущаяСтрока) – он один заменяет собой весь код выше (см. листинг 4.27). Но в этом случае для иерархического справочника будет каждый раз происходить лишнее обращение к серверу

для определения того, является открываемый элемент группой или нет. Но если это будет не иерархический справочник, а обычный, или другой объект конфигурации, документ например, этого обращения не будет.

Этот пример хорошо иллюстрирует то, что нет «рецептов на все времена» и каждый раз нужно думать и анализировать, как решить конкретную задачу наиболее эффективно.

Использование стандартного параметра формы отчета для автоматического формирования отчета при его открытии

Пользователю часто может понадобиться сформировать отчет сразу при его открытии. Эту очень удобную возможность предоставляет платформа «1С:Предприятие».

Для этого можно использовать параметр `СформироватьПриОткрытии`, поставляемый расширением формы для отчета, и передавать его в форму при открытии. Или же можно установить этот параметр в значение `Истина` в обработчике события формы `ПриСозданииНаСервере`.

Казалось бы, можно сформировать отчет и в обработчике события формы `ПриОткрытии` методом `СкомпоноватьРезультат()`, доступном в модуле формы на клиенте.

Но посмотрим, какое решение будет наиболее эффективным.

Предположим, в конфигурации существует отчет `РеестрОказанныхУслуг`, выводящий список существующих в информационной базе документов `ОказаниеУслуги`. При открытии формы отчета список документов должен формироваться автоматически, без нажатия кнопки `Сформировать`.

Для примера взят самый простой отчет, но он, тем не менее, позволит рассмотреть возможные варианты решения этой задачи.

Первый вариант решения

Чтобы программно сформировать отчет РеестрОказанныхУслуг при открытии, в форме отчета, в обработчике события ПриОткрытии, вызовем автоматическое формирование отчета.

Для этого создадим форму отчета и обработчик события формы ПриОткрытии и заполним его следующим образом (листинг 4.30).

Листинг 4.30. Обработчик события «ПриОткрытии»

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)

    СкомпоноватьРезультат();

КонецПроцедуры
```

В этом обработчике при открытии формы на клиенте отчет формируется методом `СкомпоноватьРезультат()`, поставляемым расширением формы для отчета.

Запустим «1С:Предприятие» и откроем отчет РеестрОказанныхУслуг из группы команд Отчеты (рис. 4.50).

Этот пример можно посмотреть в демонстрационной конфигурации «08 (вар. 1) Использование стандартного параметра формы отчета для автоматического формирования отчета при его открытии».

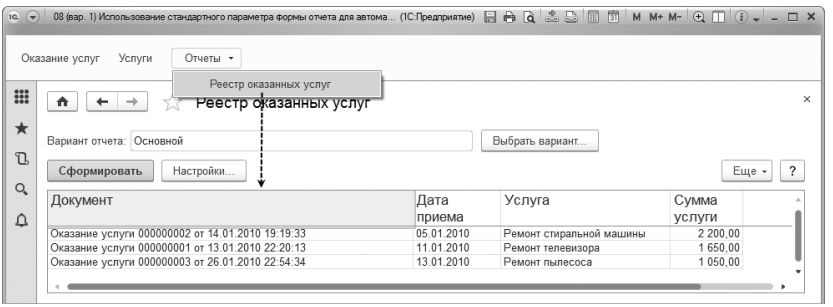


Рис. 4.50. Формирование отчета при его открытии

Отчет будет сформирован сразу при открытии формы. Вроде бы все хорошо, но в данном случае при открытии формы отчета будут сделаны *два обращения на сервер* (рис. 4.51).

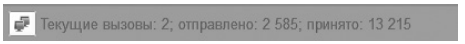


Рис. 4.51. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.52).

Один вызов происходит при открытии формы отчета, и его делает сама платформа.

А вот второго вызова, который происходит при выполнении метода `СкомпоноватьРезультат()`, можно избежать.

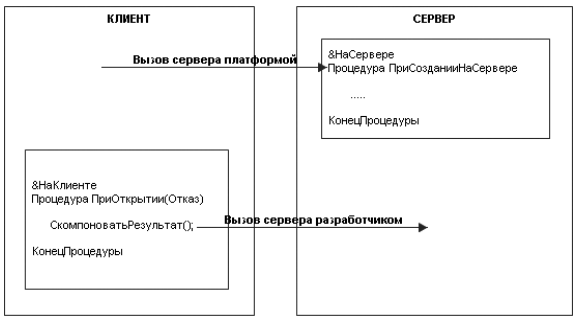


Рис. 4.52. Схема программного взаимодействия сервера и клиента

Ниже мы покажем, как это сделать.

Второй вариант решения

На самом деле можно выполнить открытие формы отчета и его формирование за один серверный вызов. Для этого нужно установить параметр формы `СформироватьПриОткрытии` в значение `Истина` в обработчике события формы ПриСозданииНаСервере.

Итак, создадим обработчик события формы отчета РеестрОказанныхУслуг – ПриСозданииНаСервере и заполним его следующим образом (листинг 4.31).

Листинг 4.31. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Параметры.СформироватьПриОткрытии = Истина;

КонецПроцедуры
```

А обработчик события ПриОткрытии удалим.

Запустим «1С:Предприятие» и откроем отчет РеестрОказанныхУслуг из группы команд Отчеты. Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, так как для открытия формы отчета и его формирования потребовался *только один вызов сервера* (рис. 4.53).

Этот пример можно посмотреть в демонстрационной конфигурации «08 (вар. 2) Использование стандартного параметра формы отчета для автоматического формирования отчета при его открытии».

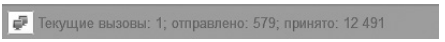


Рис. 4.53. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Третий вариант решения

Для формирования отчета в первых двух случаях мы использовали стандартную команду открытия формы отчета, автоматически помещаемую платформой в глобальные команды приложения. При этом в обоих случаях отчет всегда будет формироваться при открытии с помощью этой стандартной команды.

Теперь покажем вариант программного открытия формы отчета с параметром СформироватьПриОткрытии.

Для этого создадим общую команду СписокОказанныхУслуг и поместим ее в группу Панель навигации.Важное.

В модуле команды напишем следующий код (листинг 4.32).

Листинг 4.32. Модуль команды «СписокОказанныхУслуг»

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

    ПараметрыФормы = Новый Структура("СформироватьПриОткрытии", Истина);
    ОткрытьФорму("Отчет.РеестрОказанныхУслуг.Форма", ПараметрыФормы);

КонецПроцедуры
```

В этом обработчике мы создаем структуру ПараметрыФормы и добавляем в нее элемент СформироватьПриОткрытии со значением Истина. Затем мы открываем форму отчета РеестрОказанныхУслуг с этим параметром.

А форму отчета РеестрОказанныхУслуг можно теперь вообще удалить, поскольку она больше не нужна. А также уберем видимость у стандартной команды открытия формы отчета в командном интерфейсе конфигурации.

Этот пример можно посмотреть в демонстрационной конфигурации «08 (вар. 3) Использование стандартного параметра формы отчета для автоматического формирования отчета при его открытии».

Запустим «1С:Предприятие» и откроем отчет РеестрОказанныхУслуг из панели функций текущего раздела (рис. 4.54).

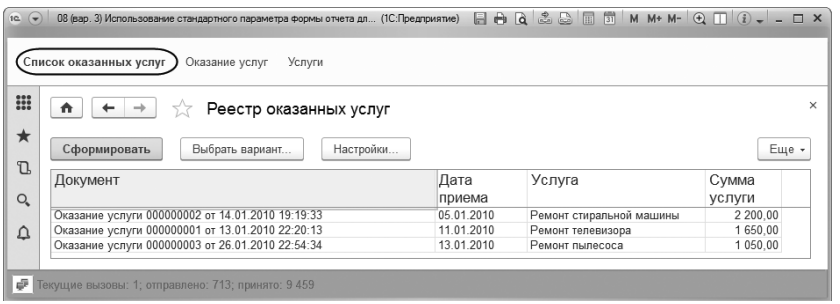


Рис. 4.54. Формирование отчета при его открытии

Отчет будет сформирован сразу при открытии формы *за один серверный вызов*.

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Резюме

Платформа «1С:Предприятие» предоставляет очень удобную возможность формировать отчет, построенный с помощью системы компоновки данных, сразу при его открытии.

Для этого *нужно использовать параметр* `СформироватьПриОткрытии`, *поставляемый расширением формы для отчета, и передавать его в форму при открытии. Или же можно установить параметр формы* `СформироватьПриОткрытии` *в значение* `Истина` *в обработчике события формы* ПриСозданииНаСервере.

При открытии формы отчета настоятельно не рекомендуется выполнять обращения к серверу, например формировать отчет, из кода модуля формы в обработчиках клиентских событий формы, таких как ПриОткрытии *и* ПриПовторномОткрытии. *Это потребует лишнего обращения к серверу и будет неэффективно.*

Таким образом, из показанных вариантов решения задачи более эффективными будут второй и третий – за счет использования параметра формы отчета `СформироватьПриОткрытии` и формирования отчета сразу при открытии *за один серверный вызов.*

Получение предопределенных значений на клиенте

Как уже не раз говорилось, работа с данными прикладных объектов на клиенте невозможна. Однако с помощью метода глобального контекста `ПредопределенноеЗначение()` на клиенте можно получать значения ссылок на предопределенные элементы справочников, планов видов характеристик, ссылки на значения перечислений и т. п.

Поэтому если нужно выполнить какое-то действие на клиенте, например открыть форму и передать в качестве параметра при ее открытии какое-то предопределенное значение, то можно получить его тут же, на клиенте, не обращаясь для этого к серверу.

Но если не задумываться о минимизации серверных вызовов с клиента и делать все «по старинке», то можно получить это предопределенное значение с сервера непосредственно перед открытием параметризированной формы.

Посмотрим, какое решение будет наиболее эффективным.

Предположим, в конфигурации существует документ ОказаниеУслуги, содержащий реквизит ВидРаботы. При заполнении соответствующего поля документа указывается, где выполнялась услуга: на дому или в офисе. Выбор нужного значения тумблера выполняется из значений перечисления ВидыРабот (ВОфисе, НаДому) – рис. 4.55.

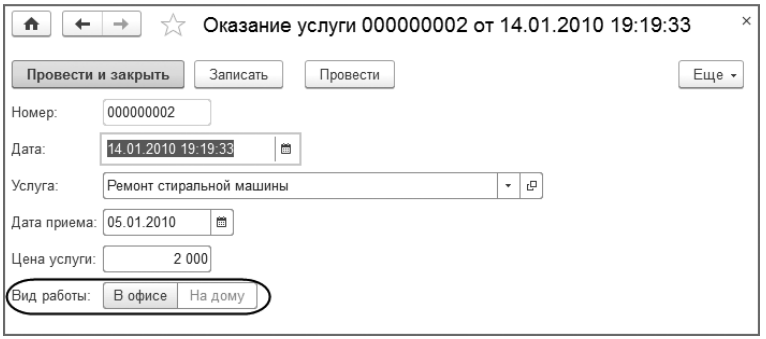


Рис. 4.55. Указание вида работы в документе «Оказание услуги»

При нажатии кнопки Список услуг на дому из списка документов ОказаниеУслуги должен открываться отчет РеестрОказанныхУслуг, выводящий список документов об оказании услуг, выполненных на дому. То есть при формировании отчета к списку услуг должен быть применен отбор по виду работы. И при этом отчет должен быть сформирован сразу же при его открытии (рис. 4.56).

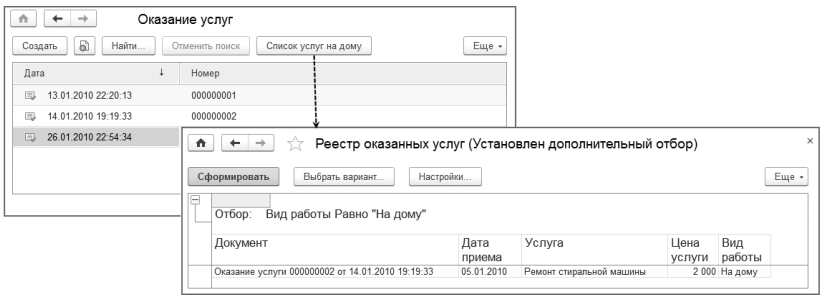


Рис. 4.56. Формирование отчета с отбором по виду работы

Первый вариант решения

Создадим форму списка документа ОказаниеУслуги и ее команду СписокУслугНаДому. Перетащим команду в командную панель формы. Обработчик команды заполним следующим образом (листинг 4.33).

Листинг 4.33. Обработчик команды «Список услуг на дому»

```
&НаКлиенте
Процедура СписокУслугНаДому(Команда)

    ОтборПоВидуРаботы = Новый Структура("ВидРаботы", ПолучитьВидРаботы());
    ПараметрыФормы = Новый Структура("Отбор, СформироватьПриОткрытии",
        ОтборПоВидуРаботы, Истина);
    ОткрытьФорму("Отчет.РеестрОказанныхУслуг.Форма", ПараметрыФормы);

КонецПроцедуры
```

В этом обработчике мы создаем структуру ОтборПоВидуРаботы, которая задает условие отбора – по виду работ, которые оказывались на дому. Для того чтобы получить ссылку на значение НаДому перечисления ВидыРабот, мы используем функцию ПолучитьВидРаботы(), которая выполняется на сервере без контекста формы (см. листинг 4.34).

Затем формируем структуру параметров формы (ПараметрыФормы), описав в ней значения двух стандартных параметров, поставляемых расширением отчета: Отбор и СформироватьПриОткрытии. Последний параметр устанавливаем в Истина, чтобы отчет формировался сразу же при его открытии. Этот вопрос мы подробно рассматривали в предыдущем примере, поэтому не будем еще раз на нем останавливаться.

И затем открываем форму отчета, передав в нее указанные параметры.

Для получения ссылки на значение перечисления ВидыРабот поместим в модуле формы функцию ПолучитьВидРаботы(), исполняющуюся на сервере без контекста формы (листинг 4.34).

Листинг 4.34. Функция «ПолучитьВидРаботы»

```
&НаСервереБезКонтекста
Функция ПолучитьВидРаботы()

    Возврат Перечисления.ВидыРабот.НаДому;

КонецФункции
```

Запустим «1С:Предприятие», откроем список документов Оказание услуг и нажмем кнопку Список услуг на дому (рис. 4.56). В результате будет открыт и сразу же сформирован отчет РеестрОказанныхУслуг, выводящий список документов об оказании услуг, выполненных на дому.

Этот пример можно посмотреть в демонстрационной конфигурации «09 (вар. 1) Получение предопределенных значений на клиенте».

Вроде бы мы все сделали правильно – открыли форму отчета с параметрами (как и было рекомендовано в предыдущих примерах), но все равно при этом были сделаны *два обращения на сервер* (рис. 4.57).

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

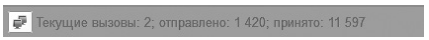


Рис. 4.57. Показатели производительности

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.58).

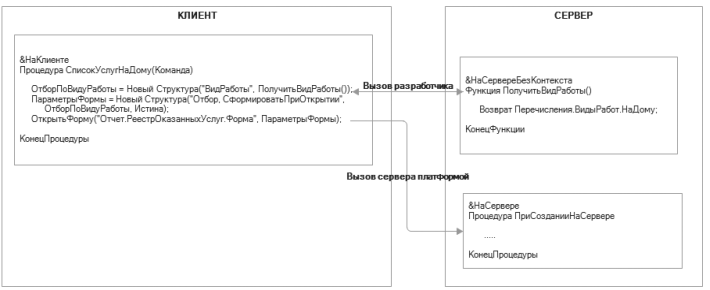


Рис. 4.58. Схема программного взаимодействия сервера и клиента

Второй вызов сервера происходит при открытии формы отчета, и его делает сама платформа.

А вот первого вызова, который происходит при получении ссылки на значение перечисления, можно избежать. Ниже мы покажем, как это сделать.

Второй вариант решения

На самом деле для получения ссылки на значение перечисления вовсе не обязательно обращаться на сервер. Это можно сделать на клиенте с помощью метода ПредопределенноеЗначение(). И затем уже передавать полученное значение в форму отчета при ее открытии.

Итак, изменим обработчик команды СписокУслугНаДому следующим образом (листинг 4.35).

Листинг 4.35. Обработчик команды «Список услуг на дому»

```
&НаКлиенте
Процедура СписокУслугНаДому(Команда)

    ВидРаботы = ПредопределенноеЗначение("Перечисление.ВидыРабот.НаДому");
    ОтборПоВидуРаботы = Новый Структура("ВидРаботы", ВидРаботы);
    ПараметрыФормы = Новый Структура("Отбор, СформироватьПриОткрытии",
        ОтборПоВидуРаботы, Истина);
    ОткрытьФорму("Отчет.РеестрОказанныхУслуг.Форма", ПараметрыФормы);

КонецПроцедуры
```

А функцию ПолучитьВидРаботы() удалим из модуля формы.

Запустим «1С:Предприятие», откроем список документов Оказание услуг и нажмем кнопку Список услуг на дому (рис. 4.56). В результате будет открыт и сразу же сформирован отчет РеестрОказанныхУслуг, выводящий список документов об оказании услуг, выполненных на дому.

Этот пример можно посмотреть в демонстрационной конфигурации «09 (вар. 2) Получение предопределенных значений на клиенте».

Поскольку ссылку на значение перечисления ВидыРабот мы получаем на клиенте, нам не понадобится лишний раз обращаться на сервер.

В результате та же функциональность будет реализована *за один серверный вызов*, и производительность прикладного решения будет выше (рис. 4.59).

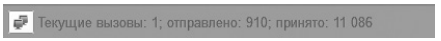


Рис. 4.59. Показатели производительности

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

Кроме того, можно открывать форму отчета РеестрОказанныхУслуг без параметров, а значения параметров `Отбор` и `СформироватьПриОткрытии` устанавливать в обработчике события формы ПриСозданииНаСервере. Но для этого, во-первых, нужно создавать форму отчета, а во-вторых, проверять в этом обработчике значение параметра `Отбор`. Так как, в общем случае, отчет может быть открыт без отбора.

Резюме

В целях минимизации серверных вызовов в тех случаях, когда на клиенте требуется получать значения ссылок на предопределенные элементы справочников, планов видов характеристик, ссылки на значения перечислений и т. п., можно сделать это с помощью метода глобального контекста `ПредопределенноеЗначение()`, не обращаясь лишний раз к серверу. При этом клиент-серверное взаимодействие не ухудшается, так как серверный вызов выполняется только при первом обращении к значению, а результат автоматически кешируется в кеше конфигурации на клиенте до следующего обновления версии конфигурации или версии платформы.

Поэтому *при выполнении каких-то действий на клиенте не нужно вызывать серверные функции для получения ссылок на предопределенные значения, значения перечислений и т. п. Это потребует лишнего обращения к серверу и будет неэффективно. Вместо этого лучше использовать метод глобального контекста* `ПредопределенноеЗначение()`.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет получения ссылки на значение перечисления *на клиенте без обращения к серверу.*

Запись данных объекта в единой транзакции за один серверный вызов

В процессе работы прикладного решения значения реквизитов объекта могут меняться не только интерактивно, но и программно. При этом *запись данных объекта в информационную базу должна выполняться в единой транзакции за один серверный вызов.*

Рассмотрим этот вопрос на примере работы с файлами и картинками.

Предположим, в форме элемента справочника Товары содержится поле, отражающее картинку товара. При нажатии кнопки Загрузить картинку, расположенной в форме документа, должен открываться диалог выбора файла картинки. Выбранная картинка должна быть отображена в форме элемента и записана в соответствующий реквизит объекта (рис. 4.60).

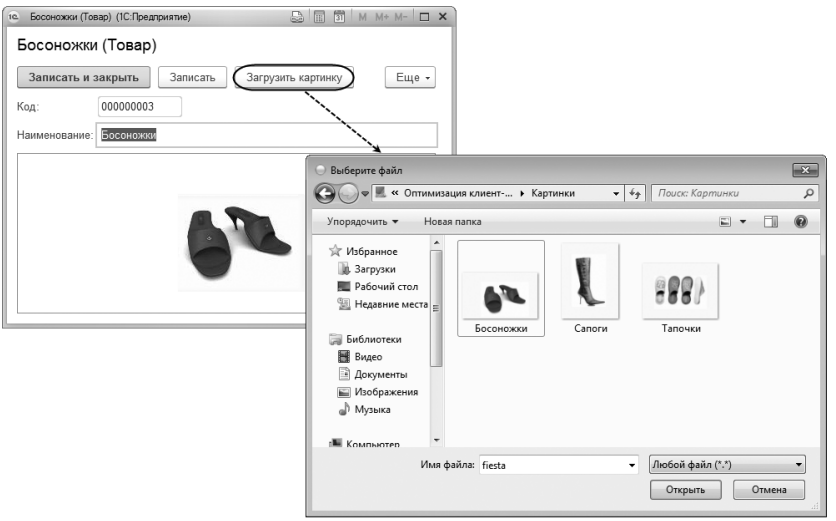


Рис. 4.60. Выбор картинки в форме товара

Сначала пользователь должен выбрать картинку товара. Единственный способ работы с файлами пользователя – интерактивно выбрать картинку и поместить ее во временное хранилище на сервере.

Затем получить ее из временного хранилища и записать в базу данных, например в реквизит объекта.

В форме картинка отображается автоматически по навигационной ссылке на временное хранилище или на реквизит объекта, содержащий эту картинку.

Когда же нужно записывать картинку из временного хранилища в соответствующий реквизит объекта? Можно записать картинку сразу после выбора, но более эффективно записать ее тогда, когда пользователь захочет сохранить объект.

Рассмотрим возможные варианты решения этой задачи.

Первый вариант решения

В данном решении запишем картинку в базу данных сразу же после ее выбора.

Итак, создадим форму элемента справочника Товары. Добавим в форму реквизит АдресКартинки (тип Строка), в котором будет храниться навигационная ссылка на реквизит справочника ДанныеФайлаКартинки. Перетащим реквизит АдресКартинки в дерево элементов формы. Установим свойства элемента: Вид – Поле картинки и ПоложениеЗаголовка – Нет.

Создадим команду формы ЗагрузитьКартинку и поместим ее в командную панель формы. При нажатии этой кнопки будут выполняться выбор файла картинки с диска, запись данных картинки в реквизит справочника ДанныеФайлаКартинки и отображение картинки в форме (рис. 4.61).

Обработчик команды ЗагрузитьКартинку заполним следующим образом (листинг 4.36).

Листинг 4.36. Обработчик команды «Загрузить картинку»

```
&НаКлиенте
Процедура ЗагрузитьКартинку(Команда)

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьКартинкуЗавершение",
        ЭтотОбъект), , , Истина , УникальныйИдентификатор);

КонецПроцедуры
```

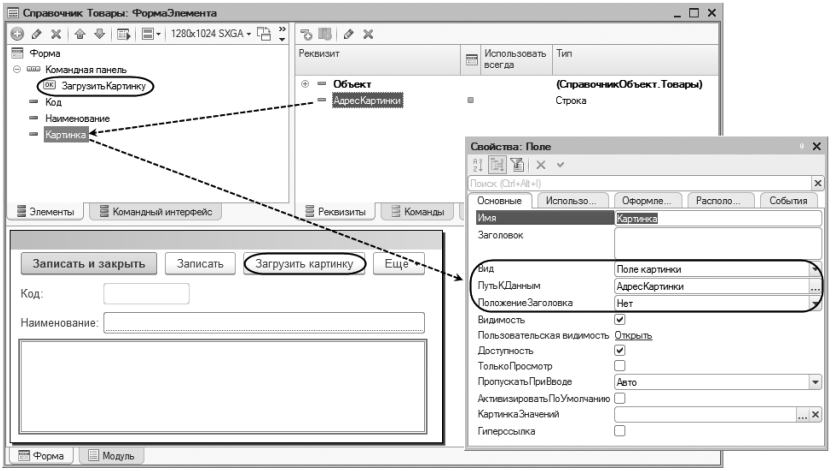


Рис. 4.61. Форма элемента справочника «Товары»

Загрузка картинки выполняется с помощью немодального метода `НачатьПомещениеФайла()`, который мы подробно рассматривали выше на стр. 642.

В первом параметре этого метода описывается процедура обработки оповещения `ЗагрузитьКартинкуЗавершение()`, в которой в случае выбора файла картинки пользователем будут выполняться действия по загрузке картинки товара (листинг 4.37).

Листинг 4.37. Обработчик оповещения «ЗагрузитьКартинкуЗавершение»

```
&НаКлиенте
Процедура ЗагрузитьКартинкуЗавершение(Результат, АдресВХранилище, ВыбранноеИмяФайла,
                                      ДополнительныеПараметры) Экспорт

    Если Результат Тогда
        Файл = Новый Файл(ВыбранноеИмяФайла);
        Объект.ИмяФайла = Файл.Имя;
        АдресКартинки = АдресВХранилище;

        ПоместитьФайлКартинки();

    КонецЕсли;

    АдресКартинки = ПолучитьНавигационнуюСсылку(Объект.Ссылка,
        "ДанныеФайлаКартинки");

КонецПроцедуры
```

В процедуру обработки оповещения в параметре АдресВХранилище передается адрес во временном хранилище, по которому был помещен файл картинки. В параметре ВыбранноеИмяФайла передается путь к выбранному файлу.

В случае успешного выбора картинки короткое имя файла сохраняется в реквизите справочника ИмяФайла, и данные файла картинки помещаются во временное хранилище. Адрес данных в этом хранилище сохраняется в реквизите формы АдресКартинки.

Затем в процедуре ПоместитьФайлКартинки() выполняется запись данных картинки в реквизит справочника ДанныеФайлаКартинки. И затем с помощью метода ПолучитьНавигационнуюССылку() реквизиту АдресКартинки присваивается навигационная ссылка на реквизит ДанныеФайлаКартинки, и картинка отображается в форме. В первом параметре метода ПолучитьНавигационнуюССылку() передается ссылка на объект Объект.Ссылка, а во втором – имя реквизита.

Контекстную серверную процедуру ПоместитьФайлКартинки() заполним следующим образом (листинг 4.38).

Листинг 4.38. Процедура «ПоместитьФайлКартинки()»

```
&НаСервере
Процедура ПоместитьФайлКартинки()

	ЭлементСправочника = РеквизитФормыВЗначение("Объект");
	ДвоичныеДанные = ПолучитьИзВременногоХранилища(АдресКартинки);
	ЭлементСправочника.ДанныеФайлаКартинки = Новый ХранилищеЗначения(
		ДвоичныеДанные, Новый СжатиеДанных());
	ЭлементСправочника.Записать();

	УдалитьИзВременногоХранилища(АдресКартинки);
	ЗначениеВРеквизитФормы(ЭлементСправочника, "Объект");

КонецПроцедуры
```

Чтобы мы могли записать объект, в этой процедуре значение основного реквизита формы Объект преобразуется в значение объекта – элемента справочника. Затем в переменной ДвоичныеДанные сохраняются данные, полученные из временного хранилища по адресу АдресКартинки. Затем создается объект ХранилищеЗначения, в него помещается значение переменной ДвоичныеДанные, и значение объекта ХранилищеЗначения сохраняется в реквизите

справочника ДанныеФайлаКартинки. Затем данные объекта – элемента справочника записываются, временное хранилище очищается, и значение измененного объекта преобразовывается обратно в значение основного реквизита формы.

Теперь осталось обеспечить чтение картинки при чтении данных уже существующего объекта. Для этого создадим обработчик события формы ПриЧтенииНаСервере и заполним его следующим образом (листинг 4.39).

Листинг 4.39. Обработчик события «ПриЧтенииНаСервере»

```
&НаСервере
Процедура ПриЧтенииНаСервере(ТекущийОбъект)

    АдресКартинки = ПолучитьНавигационнуюСсылку(ТекущийОбъект.Ссылка,
        "ДанныеФайлаКартинки");

КонецПроцедуры
```

Запустим «1С:Предприятие» и откроем форму элемента справочника Товары. Если у товара уже была выбрана картинка, она отразится в форме. Нажмем кнопку Загрузить картинку и изменим картинку товара. Выбранная картинка отобразится в поле картинки товара и будет записана в реквизит объекта ДанныеФайлаКартинки (рис. 4.60).

Этот пример можно посмотреть в демонстрационной конфигурации «10 (вар. 1) Запись данных объекта в единой транзакции за один серверный вызов».

Таким образом, мы добились нужной функциональности формы, но посмотрим теперь на показатели производительности. В результате из обработчика события нажатия кнопки Загрузить картинку (см. листинг 4.36) *будет сделано целых пять вызовов сервера!* (рис. 4.62).

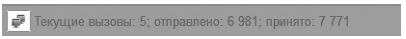


Рис. 4.62. Показатели производительности

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.63).

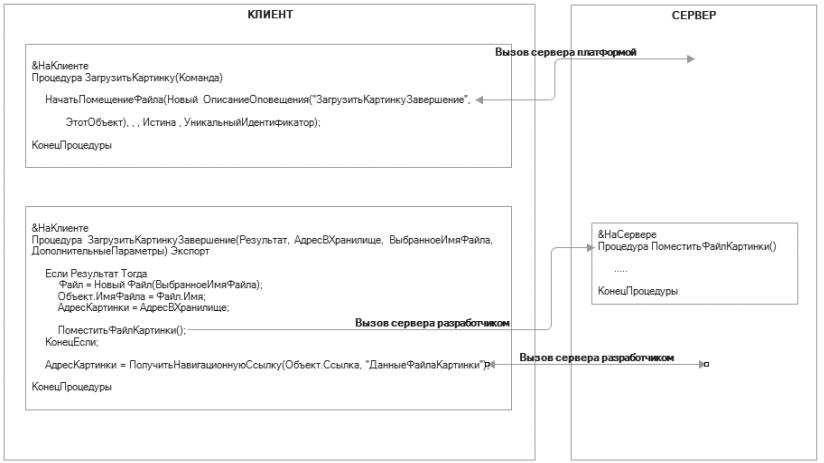


Рис. 4.63. Схема программного взаимодействия сервера и клиента

Два вызова сервера произойдут при выполнении метода `НачатьПомещениеФайла()`, и эти вызовы оправданны, так как их делает платформа. Один раз сервер вызывается при исполнении метода `НачатьПомещениеФайла()`, так как он передает картинку с клиентского компьютера во временное хранилище, а второй вызов происходит, потому что навигационная ссылка, возвращенная этим методом, сразу же помещается в реквизит формы `АдресКартинки`. Это приводит к тому, что форма обновляется, чтобы отобразить картинку по этой ссылке. Если в вызове метода `НачатьПомещениеФайла()` вместо `АдресКартинки` использовать другую произвольную переменную, а не реквизит формы, то произойдет только один серверный вызов.

Один вызов сервера будет сделан при записи картинки объекта при вызове серверной процедуры `ПоместитьФайлКартинки()`. Этого вызова можно избежать, если не записывать картинку сразу при выборе, а хранить данные картинки во временном хранилище до тех пор, пока объект не записан.

И еще два вызова произойдут при выполнении метода `ПолучитьНавигационнуюСсылку()`. Этот метод выполняет обращение к серверу при получении ссылки на реквизит. Здесь один серверный вызов происходит при получении навигационной ссылки, а второй –

потому что она сразу же присваивается реквизиту формы АдресКартинки. Этих вызовов также можно избежать, если использовать навигационную ссылку на временное хранилище для отображения картинки в форме до тех пор, пока объект не записан.

Второй вариант решения

Итак, устраним описанные выше ошибки. Для этого изменим обработчик команды ЗагрузитьКартинку следующим образом (листинг 4.40).

Листинг 4.40. Обработчик команды «Загрузить картинку»

```
&НаКлиенте
Процедура ЗагрузитьКартинку(Команда)

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьКартинкуЗавершение",
        ЭтотОбъект), , , Истина , УникальныйИдентификатор);

КонецПроцедуры
```

В этом обработчике методом НачатьПомещениеФайла() открывается диалог выбора файла картинки с диска.

В процедуре обработки оповещения ЗагрузитьКартинкуЗавершение() в случае выбора файла картинки пользователем будут выполняться действия по загрузке картинки товара (листинг 4.41).

Листинг 4.41. Обработчик оповещения «ЗагрузитьКартинкуЗавершение»

```
&НаКлиенте
Процедура ЗагрузитьКартинкуЗавершение(Результат, АдресВХранилище, ВыбранноеИмяФайла,
                                      ДополнительныеПараметры) Экспорт

    Если Результат Тогда
        Файл = Новый Файл(ВыбранноеИмяФайла);
        Объект.ИмяФайла = Файл.Имя;
        АдресКартинки = АдресВХранилище;

        Модифицированность = Истина;
    КонецЕсли;

КонецПроцедуры
```

В случае успешного выбора картинки короткое имя выбранного файла (ВыбранноеИмя) сохраняется в реквизите справочника ИмяФайла,

и данные файла картинки помещаются во временное хранилище (АдресВХранилище). Адрес данных в этом хранилище сохраняется в реквизите формы АдресКартинки.

Параметр УникальныйИдентификатор связывает эти данные хранилища с нашей формой. Когда форма будет закрыта, данные из хранилища будут автоматически удалены. Свойству формы Модифицированность присваивается значение Истина, чтобы указать, что форма уже модифицирована, так как мы выбрали картинку товара.

Как мы уже говорили, при выборе картинки данные еще не записаны в объект базы данных. Возможно, пользователь и не станет сохранять данные. В этом случае мы закроем форму, и данные временного хранилища, связанные с этой формой, будут автоматически уничтожены. Но если пользователь решит записать данные формы, то тогда нам нужно будет взять картинку из временного хранилища и сохранить ее в базе данных. Для этого создадим обработчик события формы ПередЗаписьюНаСервере и заполним его следующим образом (листинг 4.42).

Листинг 4.42. Обработчик события «ПередЗаписьюНаСервере»

```
&НаСервере
Процедура ПередЗаписьюНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

	Если ЭтоАдресВременногоХранилища(АдресКартинки) Тогда
		ДвоичныеДанные = ПолучитьИзВременногоХранилища(АдресКартинки);
		ТекущийОбъект.ДанныеФайлаКартинки = Новый ХранилищеЗначения(ДвоичныеДанные,
			Новый СжатиеДанных());

	КонецЕсли;

КонецПроцедуры
```

Это событие выполняется непосредственно перед записью объекта из формы, когда пользователь нажал Записать или Записать и закрыть.

В этой процедуре мы сохраняем значение картинки в том случае, если у нас есть ссылка на временное хранилище. В переменной ДвоичныеДанные сохраняются данные, полученные из временного хранилища по адресу АдресКартинки. Затем создается объект ХранилищеЗначения, в него помещается значение переменной ДвоичныеДанные, и значение объекта ХранилищеЗначения сохраняется в реквизите справочника ДанныеФайлаКартинки.

После записи картинки в объект нужно очистить временное хранилище и установить для реквизита формы АдресКартинки новую навигационную ссылку на реквизит объекта, хранящий данные картинки. Для этого создадим обработчик события формы ПриЗаписиНаСервере и заполним его следующим образом (листинг 4.43).

Листинг 4.43. Обработчик события «ПриЗаписиНаСервере»

```
&НаСервере
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    Если ЭтоАдресВременногоХранилища(АдресКартинки) Тогда
        УдалитьИзВременногоХранилища(АдресКартинки);

    КонецЕсли;

    АдресКартинки = ПолучитьНавигационнуюСсылку(ТекущийОбъект.Ссылка,
        "ДанныеФайлаКартинки");

КонецПроцедуры
```

Это событие выполняется после записи данных объекта, но еще до окончания транзакции записи. Так что если по каким-то причинам объект не был записан в базу, в форме останется все по-прежнему и реквизит АдресКартинки будет указывать на временное хранилище, в котором будет находиться картинка объекта.

Процедуру ПоместитьФайлКартинки() теперь можно удалить из модуля формы, так как она больше не нужна.

Запустим «1С:Предприятие» и откроем форму элемента справочника Товары. Если у товара уже была выбрана картинка, она отразится в форме. Нажмем кнопку Загрузить картинку и изменим картинку товара. Выбранная картинка отобразится в поле картинки товара и будет записана в реквизит объекта ДанныеФайлаКартинки в тот момент, когда пользователь решит записать товар.

Этот пример можно посмотреть в демонстрационной конфигурации «10 (вар. 2) Запись данных объекта в единой транзакции за один серверный вызов».

Как мы видим, функциональность прикладного решения будет практически такой же, как и в первом случае. Посмотрим теперь на показатели производительности.

В результате из обработчика события нажатия кнопки Загрузить картинку (см. листинг 4.40) *будут сделаны два вызова сервера* (рис. 4.64).

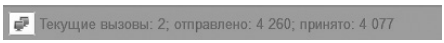


Рис. 4.64. Показатели производительности

Эти вызовы произойдут при выполнении метода НачатьПомещениеФайла(), и они оправданны, так как их делает платформа.

Посмотрим теперь, сколько вызовов сервера будет сделано при записи элемента справочника. Ведь именно в этот момент картинка окажется в базе данных. В форме товара нажмем кнопку Записать (рис. 4.65).

Рис. 4.65. Показатели производительности

При первом нажатии произойдут три серверных вызова и затем – два вызова, если мы, не закрывая формы, снова и снова нажимаем кнопку Записать.

Первый вызов происходит, когда устанавливается объектная блокировка на записываемые данные. Второй контекстный вызов сервера выполняется платформой, чтобы записать данные, отображаемые в форме. Третий – при обновлении динамического списка, открытого в основном окне программы. При последующих нажатиях кнопки Записать выполняются последние два вызова. Блокировка устанавливается только один раз и будет снята при закрытии формы.

Таким образом, видно, что при записи выполняются «обычные» вызовы сервера, которые платформа сделала бы в любом случае. То есть на запись картинки товара мы не потратили дополнительных вызовов сервера.

В результате мы *избавились от трех лишних вызовов сервера*, которые были у нас в первом случае при вызове методов ПоместитьФайлКартинки() и ПолучитьНавигационнуюСсылку().

Резюме

Запись объектов конфигурации в информационную базу должна производиться не когда придется, а в нужный момент, и не должна понижать эффективность прикладного решения.

Действия, логически объединенные в одну транзакцию (например, запись реквизитов объекта), нужно выполнять в одном серверном вызове. Это обеспечит согласованность данных объекта и избавит от лишних вызовов сервера.

При выборе картинки объекта, пока объект не записан пользователем, загруженные картинки должны быть во временном хранилище. Как только объект записывается, нужно переносить картинки из временного хранилища в реквизит и записывать синхронно с объектом (в одной транзакции) .

Что плохого, если записывать картинку в базу сразу после ее выбора? При этом будут произведены лишние вызовы сервера, ведь нет уверенности, что именно эта картинка и окажется в объекте в результате действий пользователя. Пользователь может вообще отказаться от записи объекта в базу данных. Также тут есть вопрос согласованности данных. Картинку записали, а объект – «не смогли». В результате данные могут быть несогласованными. То есть такие изменения должны выполняться в одной транзакции, за один серверный вызов.

Поэтому *не нужно записывать картинку в базу данных сразу при ее выборе, а делать это нужно непосредственно перед записью самого объекта в базу данных*.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет записи картинки объекта синхронно с самим объектом в *одной транзакции за один серверный вызов*.

Использование временного хранилища для передачи данных между формами

В процессе работы прикладного решения часто бывает нужно передавать данные из одной формы в другую. Примером такого обмена данными между формами является передача табличной части из формы документа в форму подбора и обратно.

Можно передавать данные через клиент в параметре типа ДанныеФормыКоллекция, а можно использовать для этого временное хранилище, работа с которым будет происходить на сервере.

С одной стороны, при передаче данных через клиент мы минимизируем вызовы сервера. Но объем передаваемых данных может быть при этом довольно большим. Кроме того, поскольку на клиенте присутствуют не все данные коллекций форм, а только их видимая часть (см. пример 4), то данные при передаче будут постепенно «дочитываться» с сервера.

С другой стороны, этого можно избежать, передавая данные через временное хранилище, минуя клиент, хотя для этого и потребуются дополнительные вызовы сервера.

Посмотрим, какое решение более оптимально. Рассмотрим пример.

Предположим, в табличную часть документа РасходнаяНакладная подбираются товары из справочника Товары. При нажатии кнопки Подбор, расположенной в форме документа, должна открываться форма подбора, в которую передаются строки табличной части документа. После того как пользователь произведет подбор товаров из справочника и закроет форму подбора, отобранные товары передаются обратно в табличную часть расходной накладной (рис. 4.66).

Рассмотрим возможные варианты решения этой задачи.

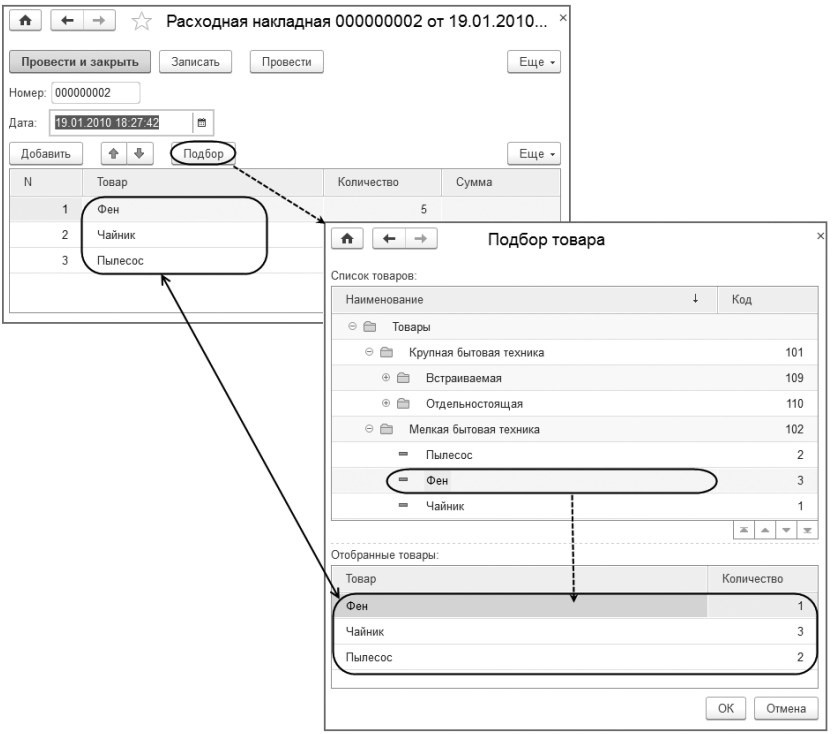


Рис. 4.66. Подбор товаров в табличную часть расходной накладной

Первый вариант решения

Итак, создадим форму документа РасходнаяНакладная. Создадим команду формы Подбор и перетащим ее в командную панель таблицы формы Товары. Обработчик команды Подбор заполним следующим образом (листинг 4.44).

Листинг 4.44. Обработчик команды «Подбор»

```
&НаКлиенте
Процедура Подбор(Команда)

    ПараметрыПодбора = Новый Структура("ТчТоваровДокумента", Объект.Товары);
    ФормаПодбора = ОткрытьФорму("Документ.РасходнаяНакладная.Форма.ФормаПодбора",
        ПараметрыПодбора, ЭтаФорма);

КонецПроцедуры
```

В этом обработчике мы создаем структуру ПараметрыФормы и добавляем в нее элемент ТчТоваровДокумента, который будет содержать табличную часть документа (Объект.Товары). Затем мы открываем форму подбора документа ФормаПодбора с параметром типа ДанныеФормыКоллекция как подчиненную форме документа.

Теперь создадим форму документа с именем ФормаПодбора с основным реквизитом СписокТоваров типа ДинамическийСписок, отражающим данные справочника Товары.

Затем создадим реквизит формы ОтобранныеТовары типа ТаблицаЗначений (с колонками Товар типа СправочникСсылка.Товары и Количество типа Число), который будет содержать список отобранных товаров и их количество.

Перетащим реквизиты СписокТоваров и ОтобранныеТовары в форму, а также зададим свойства этих таблиц формы: Заголовок, ПоложениеЗаголовка – Верх, ПоложениеКомандногоПанели – Нет. Зададим также свойство формы Заголовок – «Подбор товара» и выключим свойство Автозаголовок, ПоложениеКомандногоПанели – Низ.

Чтобы обеспечить заполнение реквизита ОтобранныеТовары данными табличной части расходной накладной, создадим обработчик события формы ПриСозданииНаСервере и заполним его следующим образом (листинг 4.45).

Листинг 4.45. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    ТчТоваровДокумента = Параметры.ТчТоваровДокумента;

    Для Каждого ТоварТч Из ТчТоваровДокумента Цикл
        Элемент = ОтобранныеТовары.Добавить();
        Элемент.Товар = ТоварТч.Товар;
        Элемент.Количество = ТоварТч.Количество;

    КонецЦикла;

КонецПроцедуры
```

В этом обработчике мы в цикле обходим коллекцию данных, содержащуюся в параметре формы ТчТоваровДокумента, и заполняем таблицу значений ОтобранныеТовары элементами этой коллекции.

Таким образом, список отобранных товаров при открытии формы подбора будет заполнен данными табличной части расходной накладной.

Затем реализуем возможность добавления товаров из списка товаров в таблицу отобранных товаров. Для этого создадим обработчик события ВыборЗначения у таблицы формы СписокТоваров и включим у этой таблицы свойство РежимВыбора (листинг 4.46).

Листинг 4.46. Обработчик события «ВыборЗначения» таблицы формы «СписокТоваров»

```
&НаКлиенте
Процедура СписокТоваровВыборЗначения(Элемент, Значение, СтандартнаяОбработка)

    СтандартнаяОбработка = Ложь;
    ДобавитьТовар(Значение);

КонецПроцедуры
```

В модуле формы поместим процедуру ДобавитьТовар(), в которой выполняется добавление товаров из списка товаров в таблицу отобранных товаров. Но заметим, что эта процедура не имеет отношения к передаче данных между формами (листинг 4.47).

Листинг 4.47. Процедура «ДобавитьТовар()»

```
&НаКлиенте
Процедура ДобавитьТовар(Товар)

    Элемент = ОтобранныеТовары.Вставить(0);
    Элемент.Товар = Товар;
    Элемент.Количество = 1;
    Элементы.ОтобранныеТовары.ТекущаяСтрока = Элемент.ПолучитьИдентификатор();

КонецПроцедуры
```

Затем, при нажатии кнопки OK, форма подбора должна закрываться, и список отобранных товаров должен передаваться обратно в табличную часть расходной накладной.

Для этого создадим команду формы подбора OK и перетащим ее в командную панель формы. Отключим свойство командной панели Автозаполнение и установим свойство ГоризонтальноеПоложение – Право. Также перетащим в командную панель формы команду Отмена из списка стандартных команд формы.

Обработчик команды ОК заполним следующим образом (листинг 4.48).

Листинг 4.48. Обработчик команды «ОК»

```
&НаКлиенте
Процедура ОК(Команда)

    ВладелецФормы.ОбработатьПодбор(ОтобранныеТовары);
    Закрыть();

КонецПроцедуры
```

В этом обработчике, перед тем как закрыть форму подбора, мы вызываем экспортируемую процедуру `ОбработатьПодбор()` владельца формы, то есть формы объекта – документа РасходнаяНакладная, и передаем в нее список отобранных товаров (листинг 4.49).

Листинг 4.49. Процедура «ОбработатьПодбор()»

```
&НаКлиенте
Процедура ОбработатьПодбор(ОтобранныеТовары) Экспорт

    Объект.Товары.Очистить();
    Для Каждого ТоварТч Из ОтобранныеТовары Цикл
        Элемент = Объект.Товары.Добавить();
        Элемент.Товар = ТоварТч.Товар;
        Элемент.Количество = ТоварТч.Количество;

    КонецЦикла;

    Модифицированность = Истина;

КонецПроцедуры
```

В этой процедуре, помещенной в модуле формы документа РасходнаяНакладная, мы очищаем табличную часть документа, затем в цикле обходим коллекцию данных, содержащуюся в параметре ОтобранныеТовары, и заполняем табличную часть документа элементами этой коллекции. И затем свойство формы Модифицированность устанавливаем в значение Истина, так как мы изменили табличную часть документа.

Таким образом, табличная часть расходной накладной при закрытии формы подбора по кнопке ОК будет заполнена списком отобранных товаров.

Запустим «1С:Предприятие», откроем документ Расходная накладная N° 3 (он содержит 1000 позиций товаров) и нажмем кнопку Подбор. После этого откроется форма подбора, в которую будут переданы строки табличной части документа.

Этот пример можно посмотреть в демонстрационной конфигурации «11 (вар. 1) Использование временного хранилища для передачи данных между формами».

Посмотрим, сколько вызовов сервера при этом произойдет и какой будет объем передаваемых данных между клиентом и сервером. Мы специально рассматриваем пример документа с большой табличной частью, так как на нем лучше сравнивать показатели производительности.

Таблица 4.3. Показатели производительности

Текущие вызовы	Отправлено	Принято
29	189 303	153 302

Мы видим, что сервер вызывается 29 раз. Один вызов происходит, чтобы открыть форму подбора, и 28 вызовов сервера тратятся на «дочитывание» строк табличной части в форму документа на клиент порциями по 35 строк.

Объем принятых данных отражает ситуацию, когда табличная часть «доехала» в форму документа на клиент, так как с клиента мы «отправляем» Объект.Товары в параметрах открываемой формы.

Объем отправленных данных отражает ситуацию, когда вся табличная часть с клиента «уехала» на сервер, так как она передается в параметре метода `ОткрытьФорму()`.

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (рис. 4.67).

После того как пользователь произведет подбор товаров из справочника (в нашем примере выбран всего один товар) и закроет форму подбора, отобранные товары и их количество будут переданы обратно в табличную часть расходной накладной (рис. 4.66).

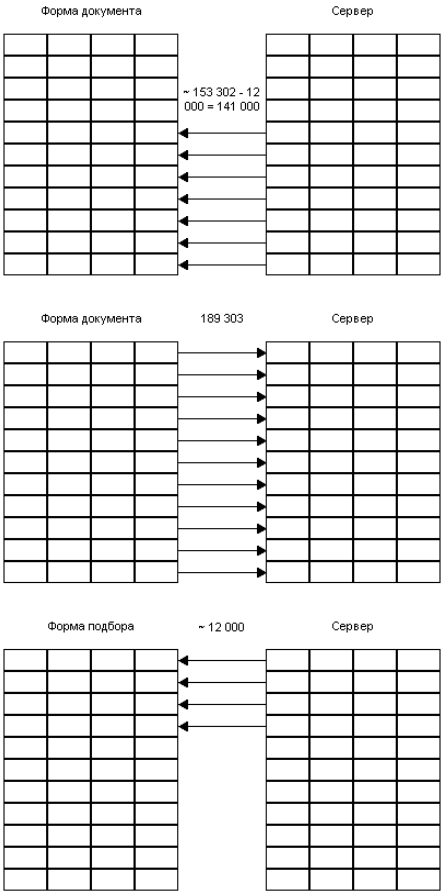


Рис. 4.67. Схема передачи данных между клиентом и сервером

Посмотрим теперь на показатели производительности.

Таблица 4.4. Показатели производительности

Текущие вызовы	Отправлено	Принято
28	11 271	97 690

28 вызовов сервера тратятся на «дочитывание» строк таблицы значений ОтобранныеТовары в форму подбора на клиент порциями по 35 строк.

Объем принятых данных отражает ситуацию, когда таблица ОтобранныеТовары «доехала» в форму подбора на клиент, так как с клиента мы вызываем процедуру `ОбработатьПодбор()`, в которую передаем эту таблицу в качестве параметра.

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (рис. 4.68).

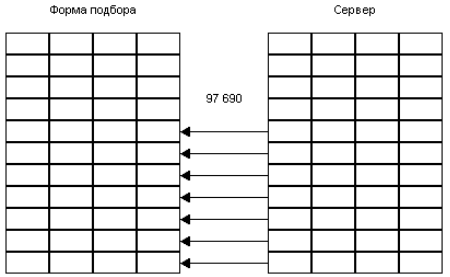


Рис. 4.68. Схема передачи данных между клиентом и сервером

Может возникнуть вопрос: «Почему в первой таблице, когда в форму документа дочитывалась табличная часть, было передано 153 302, а тут, когда те же данные дочитываются в форму подбора, передано всего 97 690?»

Дело в том, что в первой таблице дочитывалась именно табличная часть. Во второй таблице дочитывалась не табличная часть, а таблица значений. Таблица значений устроена проще, чем табличная часть, поэтому она сериализуется лучше (объем данных меньше). Кроме того, в первом примере табличная часть содержит три колонки, а таблица значений во втором случае содержит всего две колонки. За счет этого во втором случае и получается меньший объем принятых данных.

В результате мы добились нужной функциональности формы, но при большом массиве передаваемых данных такой вариант обмена данными между формами будет неоптимальным.

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.69).

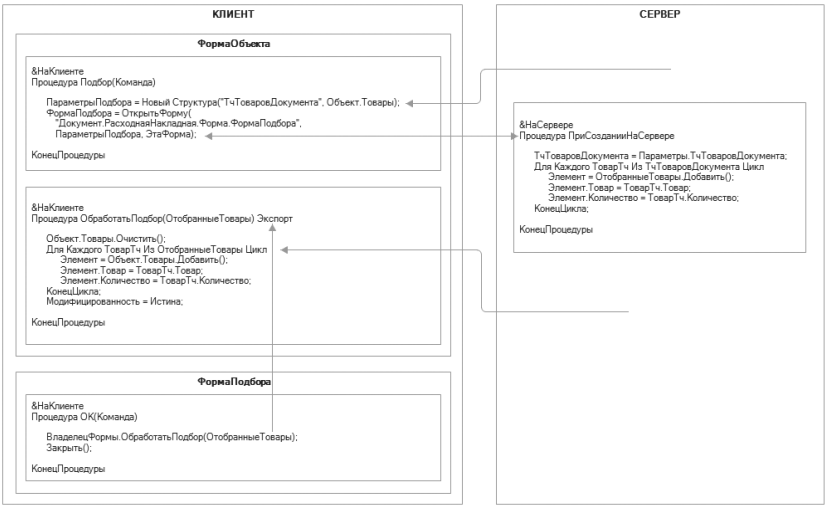


Рис. 4.69. Схема программного взаимодействия сервера и клиента

Как мы видим, при открытии формы подбора платформа делает вызов сервера.

Но при этом в форму подбора передается параметр `ТчТоваровДокумента`, содержащий данные табличной части формы объекта. Все строки табличной части сначала «приедут» с сервера, так как на клиенте присутствует только видимая часть этих строк (см. пример 4). Затем все они «поедут» на сервер при открытии формы подбора.

При передаче отобранных товаров обратно в форму документа в параметре процедуры `ОбработатьПодбор()` все данные таблицы значений `ОтобранныеТовары` формы подбора будут «дочитаны» с сервера на клиент.

> Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме. Но мы будем изучать стандартную ситуацию, когда при открытии формы происходит один вызов сервера.

Таким образом, количество вызовов сервера будет расти пропорционально количеству строк табличной части, так как строки «дочитываются» порциями по 35 строк, а также потенциально большой объем данных будет передаваться с сервера на клиент и обратно, что нехорошо, особенно в случае медленного соединения.

Рассмотрим теперь другой вариант.

Второй вариант решения

На самом деле более эффективно передавать данные между формами через временное хранилище, работа с которым будет происходить на сервере. Ведь основная масса строк табличной части находится на сервере, а не на клиенте, и поэтому логично было бы передать их другой форме прямо с сервера, не «таская» их на клиент.

Для этого перед открытием формы подбора нужно сначала записать данные табличной части документа во временное хранилище, а в форме подбора прочитать его и загрузить в список отобранных товаров.

Изменим обработчик команды Подбор следующим образом (листинг 4.50).

Листинг 4.50. Обработчик команды «Подбор»

```
&НаКлиенте
Процедура Подбор(Команда)

    АдресТоваровВХранилище = ПоместитьТоварыВХранилище();
    ПараметрыПодбора = Новый Структура("АдресТоваровДокумента", АдресТоваровВХранилище);
    ФормаПодбора = ОткрытьФорму("Документ.РасходнаяНакладная.Форма.ФормаПодбора",
        ПараметрыПодбора, ЭтаФорма);

КонецПроцедуры
```

В этом обработчике в функции `ПоместитьТоварыВХранилище()` мы помещаем данные, выгруженные из табличной части документа, во временное хранилище. Эта функция возвращает адрес данных в хранилище – `АдресТоваровВХранилище`. Затем открываем форму подбора с параметром `АдресТоваровДокумента`, содержащим этот адрес.

Поместим в модуле формы документа функцию ПоместитьТоварыВХранилище(), выполняющуюся на сервере с контекстом формы (листинг 4.51).

Листинг 4.51. Функция «ПоместитьТоварыВХранилище()»

```
&НаСервере
Функция ПоместитьТоварыВХранилище()

    Возврат ПоместитьВоВременноеХранилище(Объект.Товары.Выгрузить( ,"Товар,Количество"),
        УникальныйИдентификатор);

КонецФункции
```

В самом начале модуля формы документа мы определяем клиентскую переменную АдресТоваровВХранилище, чтобы потом получить по этому адресу список отобранных товаров из формы подбора (листинг 4.52).

Листинг 4.52. Определение переменной «АдресТоваровВХранилище»

```
&НаКлиенте
Перем АдресТоваровВХранилище;
```

Теперь создадим в форме подбора строковый реквизит АдресТоваровДокумента для хранения адреса товаров в хранилище. Заполнять его будем в обработчике события формы ПриСозданииНаСервере (листинг 4.53).

Листинг 4.53. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    АдресТоваровДокумента = Параметры.АдресТоваровДокумента;
    ОтобранныеТовары.Загрузить(ПолучитьИзВременногоХранилища(АдресТоваровДокумента));

КонецПроцедуры
```

В этом обработчике мы сохраняем в реквизите АдресТоваровДокумента адрес товаров во временном хранилище, получаем из хранилища данные по этому адресу и загружаем их в таблицу значений ОтобранныеТовары.

Таким образом, список отобранных товаров при открытии формы подбора будет заполнен данными табличной части расходной накладной.

Теперь при закрытии формы подбора по нажатию кнопки OK нужно записать список отобранных товаров во временное хранилище, а в форме документа прочитать его и загрузить обратно в табличную часть.

Изменим обработчик команды OK следующим образом (листинг 4.54).

Листинг 4.54. Обработчик команды «OK»

```
&НаКлиенте
Процедура ОК(Команда)

    ЗаписатьПодборВХранилище();
    ВладелецФормы.ОбработатьПодбор();

    Закрыть();

КонецПроцедуры
```

Поместим в модуле формы подбора процедуру `ЗаписатьПодборВХранилище()`, выполняющуюся на сервере с контекстом формы, для записи данных таблицы значений `ОтобранныеТовары` во временное хранилище по адресу `АдресТоваровДокумента` (листинг 4.55).

Листинг 4.55. Процедура «ЗаписатьПодборВХранилище()»

```
&НаСервере
Процедура ЗаписатьПодборВХранилище()

    ПоместитьВоВременноеХранилище(ОтобранныеТовары.Выгрузить(), АдресТоваровДокумента);

КонецПроцедуры
```

Процедуру `ОбработатьПодбор()` владельца формы, то есть формы объекта – документа `РасходнаяНакладная`, изменим следующим образом (листинг 4.56).

Листинг 4.56. Процедура «ОбработатьПодбор()»

```
&НаКлиенте
Процедура ОбработатьПодбор() Экспорт

    ПолучитьТоварыИзХранилища(АдресТоваровВХранилище);
    Модифицированность = Истина;

КонецПроцедуры
```

Поместим в модуле формы документа процедуру ПолучитьТоварыИзХранилища(), выполняющуюся на сервере с контекстом формы, для чтения данных из временного хранилища. В процедуру мы передаем адрес этих данных в хранилище, хранящийся в клиентской переменной АдресТоваровВХранилище (листинг 4.57).

Листинг 4.57. Процедура «ПолучитьТоварыИзХранилища()»

```
&НаСервере
Процедура ПолучитьТоварыИзХранилища(АдресТоваровВХранилище)

    Объект.Товары.Загрузить(ПолучитьИзВременногоХранилища(АдресТоваровВХранилище));

КонецПроцедуры
```

В этой процедуре мы получаем товары из временного хранилища по адресу АдресТоваровВХранилище и загружаем их в табличную часть документа.

Таким образом, табличная часть расходной накладной при закрытии формы подбора по кнопке OK будет заполнена списком отобранных товаров.

Этот пример можно посмотреть в демонстрационной конфигурации «11 (вар. 2) Использование временного хранилища для передачи данных между формами».

Запустим «1С:Предприятие», откроем документ Расходная накладная N° 3 (он содержит 1000 позиций товаров) и нажмем кнопку Подбор. После этого строки табличной части документа будут помещены во временное хранилище. Затем откроется форма подбора. При ее открытии данные из хранилища будут прочитаны и загружены в список отобранных товаров.

Посмотрим, сколько вызовов сервера при этом произойдет и какой будет объем передаваемых данных между клиентом и сервером. Мы специально рассматриваем пример документа с большой табличной частью, так как на нем лучше сравнивать показатели производительности.

Таблица 4.5. Показатели производительности

Текущие вызовы	Отправлено	Принято
2	2 353	12 536

Мы видим, что сервер вызывается 2 раза. Один вызов происходит при записи данных табличной части в хранилище, а второй – при открытии формы подбора методом `ОткрытьФорму()`.

Объем принятых данных отражает ситуацию, когда на клиент «приезжает» форма и видимые 35 строк табличной части.

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (первый вариант решения для сравнения приведен слева), рис. 4.70.

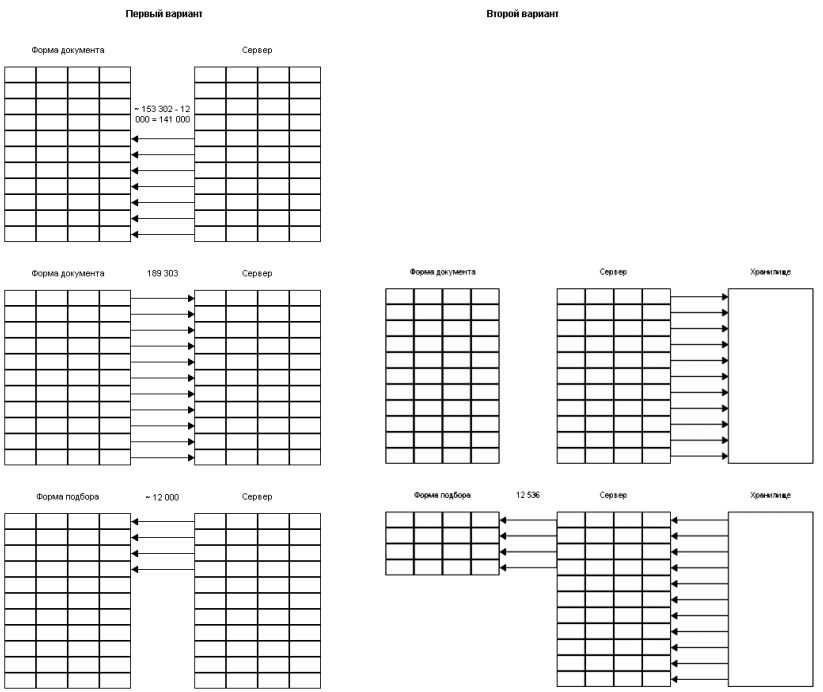


Рис. 4.70. Схема передачи данных между клиентом и сервером

После того как пользователь произведет подбор товаров из справочника (в нашем примере выбран всего один товар) и закроет форму подбора, отобранные товары и их количество будут помещены во временное хранилище. В форме документа в процедуре

ОбработатьПодбор() данные из хранилища будут прочитаны и загружены обратно в табличную часть.

Посмотрим теперь на показатели производительности.

Таблица 4.6. Показатели производительности

Текущие вызовы	Отправлено	Принято
2	5 196	182 538

Здесь один вызов сервера тратится на помещение таблицы значений ОтобранныеТовары из формы подбора в хранилище, а второй – на чтение этих данных из хранилища в форме документа.

Объем принятых данных отражает ситуацию, когда вся табличная часть «приезжает» в форму документа на клиент, так как мы ее заново загрузили на сервере из временного хранилища.

Для наглядности рассмотрим схему передачи данных между клиентом и сервером (первый вариант решения для сравнения приведен слева), рис. 4.71.

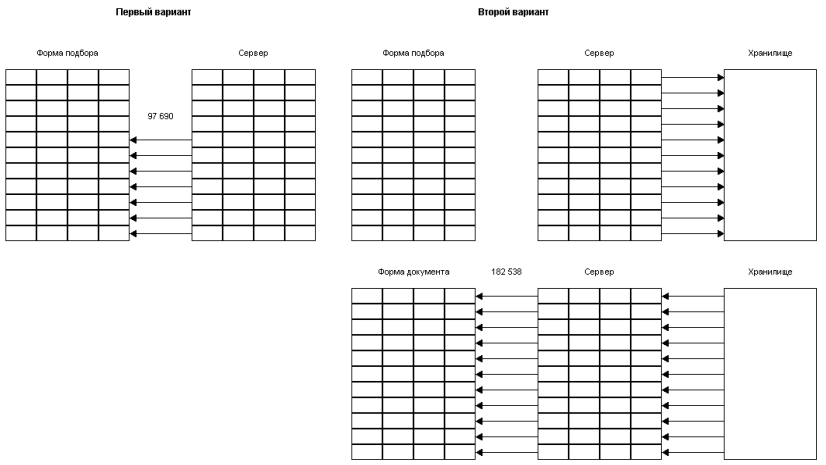


Рис. 4.71. Схема передачи данных между клиентом и сервером

Объем принятых данных близок к аналогичным показателям в первом варианте. И, действительно, все будет похоже. Здесь мы получаем всю табличную часть с сервера, и это занимает 182 538. А в первом примере мы, наоборот, передавали всю табличную часть на сервер (рис. 4.70). И там это заняло 189 303. В принципе, значения одинаковые, разница объясняется разными направлениями движения данных (может использоваться разная степень сжатия данных) и нюансами кеширования.

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.72).

Рис. 4.72. Схема программного взаимодействия сервера и клиента

При открытии формы подбора из процедуры `Подбор()` один вызов сервера произойдет при записи списка товаров табличной части документа во временное хранилище, а второй вызов – собственно при открытии формы подбора.

При закрытии формы подбора из процедуры `ОК()` один вызов сервера произойдет при записи списка отобранных товаров во временное хранилище, а второй

Заметим, что при первом открытии формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации о форме.

вызов – из процедуры формы документа `ОбработатьПодбор()` при чтении отобранных товаров из временного хранилища и загрузке их в табличную часть документа.

Итого 4 контекстных вызова сервера вместо 57 (в первом варианте)!

По объему передаваемых данных в первом случае получается: 153 302 + 189 303 + 97 690 = 440 295.

Во втором случае получается: 12 536 + 182 538 = 195 074.

То есть разница в объеме передаваемых данных больше чем в два раза!

Что показательно, в первом случае количество серверных вызовов будет увеличиваться пропорционально увеличению табличной части, а во втором варианте количество серверных вызовов будет постоянным независимо от объема табличной части. Это тоже важный момент.

Таким образом, мы видим, что при большом массиве передаваемых данных такой вариант обмена данными между формами будет более оптимальным.

Резюме

При оптимизации клиент-серверного взаимодействия нужно стремиться не только минимизировать вызовы сервера, но и следить за объемом передаваемых данных между сервером и клиентом.

Если можно лишний раз не передавать данные между клиентом и сервером, лучше их не передавать.

Поэтому *при передаче данных между формой объекта и формой подбора рекомендуется использовать временное хранилище, чтение и запись которого должны выполняться на сервере.*

Потенциально большой массив данных не рекомендуется передавать в качестве параметра формы подбора.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет передачи данных между формами, минуя клиент, через временное хранилище.

Реализация пересчета данных объекта в модуле объекта или в модуле формы в зависимости от логики объекта

Для реализации той или иной функциональности объектов конфигурации часто бывает нужно выполнить пересчет данных объекта, содержащегося в форме. Сделать это можно как в контекстной серверной процедуре модуля формы, так и в экспортируемой процедуре модуля объекта.

В первом случае производится пересчет непосредственно данных формы без получения объекта, а при записи объекта из формы эти данные конвертируются в объект и записываются.

Во втором случае предварительно нужно получить объект, преобразовав данные формы в объект, произвести пересчет и выполнить обратное преобразование объекта в данные формы. На это требуется дополнительное время, но зато процедура, инкапсулированная в объекте, может вызываться из других мест конфигурации как отдельный метод этого объекта.

Когда же нужно помещать процедуру пересчета данных объекта в модуле формы, а когда в модуле объекта? Однозначного ответа тут нет. Решение зависит от конкретной задачи. Рассмотрим пример.

Предположим, в конфигурации существует справочник Клиенты. В нем для каждого клиента хранится процент скидки, предоставляемый ему при оказании услуг. В форме документа об оказании услуг содержатся услуга, цена услуги, клиент, которому она оказывается, сумма скидки и сумма услуги. При выборе клиента нужно выяснить, предоставляется ли ему скидка, и рассчитать сумму скидки как процент от цены услуги. При этом сумма услуги рассчитывается как цена услуги за вычетом суммы скидки (рис. 4.73).

Таким образом, в тот момент, когда пользователь изменит поле Клиент или поле ЦенаУслуги документа ОказаниеУслуги, поля СуммаСкидки и СуммаУслуги нужно рассчитать по описанному выше алгоритму.

Рассмотрим возможные варианты решения этой задачи.

Рис. 4.73. Пересчет данных в документе при изменении клиента

Первый вариант решения

В данном решении поместим процедуру для начисления скидки клиенту в модуле формы документа ОказаниеУслуги.

Чтобы обеспечить перерасчет суммы скидки и суммы услуги при изменении ее цены и выборе клиента, создадим форму документа ОказаниеУслуги и обработчик события ПриИзменении для поля формы Клиент, ссылающегося на справочник Клиенты (листинг 4.58).

Листинг 4.58. Процедура «КлиентПриИзменении()»

```
&НаКлиенте
Процедура КлиентПриИзменении(Элемент)

    НачислитьСкидку();

КонецПроцедуры
```

В этом обработчике мы вызываем контекстную серверную процедуру для начисления скидки клиенту, расположенную в модуле формы (листинг 4.59).

Листинг 4.59. Процедура «НачислитьСкидку()»

```
&НаСервере
Процедура НачислитьСкидку()

    Объект.СуммаСкидки = Объект.ЦенаУслуги * Объект.Клиент.Скидка / 100;
    Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

КонецПроцедуры
```

В этой процедуре мы получаем значение реквизита Скидка по ссылке на текущий элемент справочника Клиенты (`Объект.Клиент`). Как показывалось во втором варианте первого примера, более эффективно получать значение реквизита запросом, а не через точку от ссылки, но мы для упрощения сейчас сделаем так.

И затем пересчитываем поля СуммаСкидки и СуммаУслуги документа ОказаниеУслуги по заданному алгоритму.

Теперь обеспечим пересчет суммы услуги при изменении цены услуги. Для этого создадим обработчик события ПриИзменении для поля формы ЦенаУслуги (листинг 4.60).

Листинг 4.60. Процедура «ЦенаУслугиПриИзменении()»

```
&НаКлиенте
Процедура ЦенаУслугиПриИзменении(Элемент)

    Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

КонецПроцедуры
```

Включим для полей формы СуммаСкидки и СуммаУслуги свойство ТолькоПросмотр, так как они являются расчетными.

Этот пример можно посмотреть в демонстрационной конфигурации «12 (вар. 1) Реализация пересчета данных объекта в модуле объекта или в модуле формы в зависимости от логики объекта».

Запустим «1С:Предприятие», создадим документ об оказании услуги, выберем услугу и внесем цену услуги. При этом сумма услуги становится равной цене услуги. Затем выберем клиента. После этого, если в списке клиентов указан процент скидки, предоставляемой клиенту, сумма скидки рассчитывается как этот процент от цены

услуги. При этом сумма услуги рассчитывается как цена услуги за вычетом суммы скидки (рис. 4.73).

Посмотрим теперь на показатели производительности. При выборе клиента *произойдет один вызов сервера* (рис. 4.74).

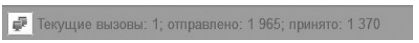


Рис. 4.74. Показатели производительности

Заметим, что при первом выборе клиента может быть лишний серверный вызов за счет кеширования платформой необходимой информации в списке выбора. Но мы будем изучать стандартную ситуацию, когда при выборе происходит один вызов сервера.

Этот вызов происходит при выполнении контекстной серверной процедуры `НачислитьСкидку()`.

Итак, мы поместили процедуру для начисления скидки клиенту в модуле формы документа ОказаниеУслуги. После записи или проведения документа из формы пересчитанные данные формы будут преобразованы в объект и записаны. Таким образом, в данном решении *мы посчитали процедуру начисления скидки «частным» делом этой формы*. Снаружи данная функциональность формы не видна.

Второй вариант решения

На самом деле может быть более логично поместить процедуру для начисления скидки клиенту в модуле объекта. Потому что начисление скидки – это часть прикладной логики самого документа ОказаниеУслуги.

Изменим процедуру `НачислитьСкидку()` в модуле формы документа следующим образом (листинг 4.61).

Листинг 4.61. Процедура «НачислитьСкидку()»

```
&НаСервере
Процедура НачислитьСкидку()

    Документ = РеквизитФормыВЗначение("Объект");
    Документ.НачислитьСкидкуКлиенту();
    ЗначениеВРеквизитФормы(Документ, "Объект");

КонецПроцедуры
```

В этой процедуре мы сначала получаем объект, отображающийся в форме. Для этого преобразуем основной реквизит формы в объект – документ ОказаниеУслуги. Затем выполняем экспортируемый метод объекта НачислитьСкидкуКлиенту() и выполняем обратное преобразование объекта в реквизит формы Объект.

Затем откроем модуль объекта и поместим в нем процедуру для начисления скидки клиенту (листинг 4.62).

Листинг 4.62. Процедура «НачислитьСкидкуКлиенту()»

```
Процедура НачислитьСкидкуКлиенту() Экспорт

    СуммаСкидки = ЦенаУслуги * Клиент.Скидка / 100;
    СуммаУслуги = ЦенаУслуги - СуммаСкидки;

КонецПроцедуры
```

В этой процедуре мы реализуем тот же алгоритм пересчета, что и в предыдущем варианте, но только пересчитываем не данные формы, а данные самого объекта. Эти изменения после преобразования объекта в основной реквизит формы отобразятся в форме документа.

Этот пример можно посмотреть в демонстрационной конфигурации «12 (вар. 2) Реализация пересчета данных объекта в модуле объекта или в модуле формы в зависимости от логики объекта».

Запустим «1С:Предприятие», создадим документ об оказании услуги, выберем услугу и внесем цену услуги. При этом сумма услуги становится равной цене услуги. Затем выберем клиента. После этого, если в списке клиентов указан процент скидки, предоставляемой клиенту, сумма скидки рассчитывается как этот процент от цены услуги. При этом сумма услуги рассчитывается как цена услуги за вычетом суммы скидки (рис. 4.73).

Посмотрим теперь на показатели производительности. При выборе клиента *произойдет один вызов сервера* (рис. 4.75).

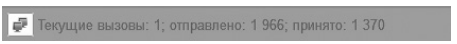


Рис. 4.75. Показатели производительности

Этот вызов происходит при выполнении контекстной серверной процедуры модуля формы НачислитьСкидку(), а из нее вызывается экспортируемый метод документа НачислитьСкидкуКлиенту().

Заметим, что здесь, так же как и в предыдущем варианте, при первом выборе клиента может быть лишний серверный вызов за счет кеширования платформой необходимой информации в списке выбора.

Итак, мы поместили процедуру для начисления скидки клиенту в модуле объекта. Тем самым мы предоставили доступ к данному методу документа ОказаниеУслуги из других мест конфигурации: форм, обработок и т. д. Таким образом, в данном решении *мы посчитали процедуру начисления скидки частью прикладной логики объекта*.

Данная функциональность документа доступна, например, из формы списка документов об оказании услуг. То есть мы можем изменить клиенту процент скидки, затем выделить документ об оказании услуги и пересчитать по нему скидку, не открывая документа.

Для этого создадим форму списка документа ОказаниеУслуги и ее команду НачислитьСкидку. Перетащим команду в командную панель формы. Обработчик команды НачислитьСкидку заполним следующим образом (листинг 4.63).

Листинг 4.63. Обработчик команды «Начислить скидку»

```
&НаКлиенте
Процедура НачислитьСкидку(Команда)

    НачислитьСкидкуНаСервере(Элементы.Список.ТекущаяСтрока);

КонецПроцедуры
```

В этом обработчике мы вызываем внеконтекстную серверную процедуру модуля формы НачислитьСкидкуНаСервере() и передаем в нее ссылку на текущий документ (листинг 4.64).

Листинг 4.64. Процедура «НачислитьСкидкуНаСервере»

```
&НаСервереБезКонтекста
Процедура НачислитьСкидкуНаСервере(СсылкаНаДокумент)

	Документ = СсылкаНаДокумент.ПолучитьОбъект();
	Документ.НачислитьСкидкуКлиенту();
	Документ.Записать();

КонецПроцедуры
```

В этой процедуре по переданной ссылке на документ мы получаем объект, выполняем экспортируемый метод объекта `НачислитьСкидкуКлиенту()` и записываем документ.

Запустим «1С:Предприятие», откроем список клиентов и изменим процент скидки, предоставляемой какому-либо клиенту. Затем откроем список документов об оказании услуг, выделим документ для данного клиента и нажмем кнопку Начислить скидку. Сумма скидки и сумма услуги в документе пересчитаются по заданному алгоритму (рис. 4.76).

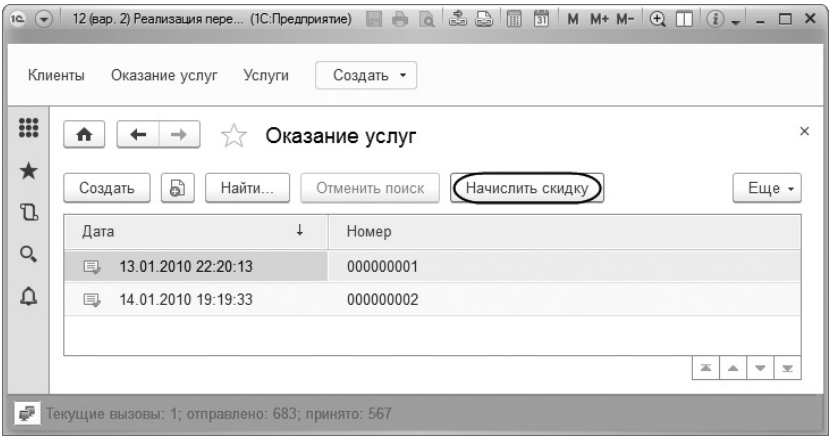


Рис. 4.76. Начисление скидки клиенту из формы списка документов

Можно организовать начисление скидки для документов в цикле и т. д.

Резюме

Пересчитывать данные объекта можно как в контекстной серверной процедуре модуля формы объекта, так и в экспортируемой процедуре модуля объекта.

Когда процедура пересчета является частью прикладной логики объекта, нужно реализовывать процедуру в модуле объекта как его экспортируемый метод.

Когда процедура пересчета отражает функциональность конкретной формы, можно реализовывать ее в модуле этой формы.

Таким образом, из показанных вариантов решения задачи более эффективным будет второй – за счет реализации пересчета данных в экспортируемом методе объекта. Этот метод будет доступен из других мест конфигурации: форм, обработок и т. д.

Изменение оформительских свойств элементов формы, не требующих обращения к серверу

Для реализации той или иной функциональности формы часто бывает нужно изменять свойства ее элементов (например, менять их оформление) в зависимости от конкретных действий пользователя.

Первое, что приходит в голову, – использовать для этого обработчик события элементов формы ПриИзменении и программно изменять требуемые свойства на клиенте.

Но нужно учитывать, что если свойства, влияющие на размер элементов формы, меняются программно (например, свойство Шрифт), то форма должна полностью «пересчитать» расположение полей. Это выполняется на сервере и требует определенных затрат времени.

В платформе «1С:Предприятие» начиная с версии 8.3.7 реализован новый механизм расстановки элементов формы. В результате реализованы новые возможности размещения элементов в форме, изменение большинства оформительских свойств элементов формы на клиенте больше не требует обращения к серверу.

Поэтому при программном изменении на клиенте желательно использовать свойства элементов формы, изменение которых не приводит к обращениям на сервер. Например, вместо свойства Шрифт можно использовать свойства ЦветФона или ЦветТекста.

Если же, в зависимости от действий пользователя, требуется изменять именно шрифт элемента, то для этого можно использовать условное оформление. При этом изменение шрифта элемента формы с помощью условного оформления не влияет на размер элемента и, соответственно, не приводит к обращению на сервер.

Какой же вариант предпочтительнее? Рассмотрим пример.

Предположим, в форме документа об оказании услуг содержатся цена услуги, флажок скидки, сумма скидки и сумма услуги. В случае установки флажка Скидка поле СуммаСкидки должно быть выделено жирным шрифтом и зеленым цветом текста. При снятии флажка Скидка поле СуммаСкидки должно обнуляться и выводиться обычным шрифтом и красным цветом текста (рис. 4.77).

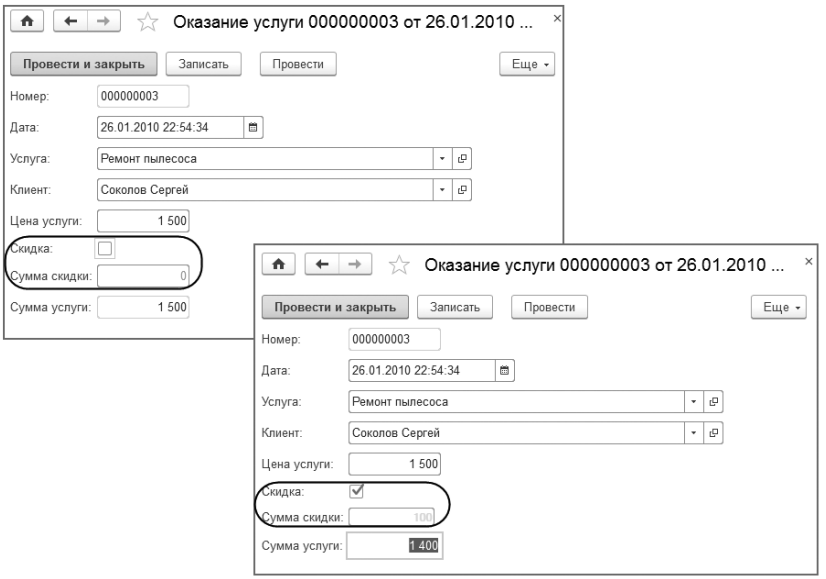


Рис. 4.77. Оформление поля «Сумма скидки» в зависимости от состояния флажка «Скидка»

Кроме того, при изменении цены услуги и суммы скидки, в зависимости от состояния флажка Скидка, нужно рассчитать сумму услуги, равную цене услуги за вычетом суммы скидки. Сразу оговоримся, что прямого отношения к теме примера эта функциональность не имеет.

Первый вариант решения

Чтобы реализовать поставленную задачу, создадим форму документа ОказаниеУслуги и обработчик события ПриИзменении для поля формы Скидка (листинг 4.65).

Листинг 4.65. Процедура «СкидкаПриИзменении()»

```
&НаКлиенте
Процедура СкидкаПриИзменении(Элемент)

    Если Объект.Скидка Тогда
        Элементы.СуммаСкидки.Шрифт = Новый Шрифт(,, Истина);
        Элементы.СуммаСкидки.ЦветТекста = WebЦвета.ЗеленаяЛужайка;

        Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

    Иначе
        Элементы.СуммаСкидки.Шрифт = Новый Шрифт();
        Элементы.СуммаСкидки.ЦветТекста = WebЦвета.Коралловый;

        Объект.СуммаСкидки = 0;
        Объект.СуммаУслуги = Объект.ЦенаУслуги;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы анализируем значение реквизита документа Скидка типа Булево и в зависимости от него устанавливаем цвет и жирность шрифта для поля формы СуммаСкидки. Новый шрифт поля мы создаем конструктором – Новый Шрифт(), при создании которого в третьем параметре указывается жирность.

Кроме того, при изменении флажка Скидка мы производим пересчет суммы услуги в соответствии с заданным алгоритмом.

Теперь обеспечим пересчет суммы услуги при изменении цены услуги и суммы скидки, в зависимости от состояния флажка Скидка.

Создадим обработчик события ПриИзменении для поля формы ЦенаУслуги и заполним его следующим образом (листинг 4.66).

Листинг 4.66. Процедура «ЦенаУслугиПриИзменении()»

```
&НаКлиенте
Процедура ЦенаУслугиПриИзменении(Элемент)

    Если Объект.Скидка Тогда
        Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

    Иначе
        Объект.СуммаУслуги = Объект.ЦенаУслуги;

    КонецЕсли;

КонецПроцедуры
```

Затем создадим обработчик события ПриИзменении для поля формы СуммаСкидки и заполним его следующим образом (листинг 4.67).

Листинг 4.67. Процедура «СуммаСкидкиПриИзменении()»

```
&НаКлиенте
Процедура СуммаСкидкиПриИзменении(Элемент)

    Если Объект.Скидка Тогда
        Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;
    КонецЕсли;

КонецПроцедуры
```

Алгоритм пересчета суммы услуги мы выполняем на клиенте, так как он небольшой и быстро выполняется из клиентской процедуры при изменении цены услуги и суммы скидки. Здесь нет ничего принципиально нового, так как этот вопрос мы уже рассматривали во втором варианте третьего примера.

Включим для поля формы СуммаУслуги свойство ТолькоПросмотр, так как оно является расчетным.

И в заключение установим оформление поля СуммаСкидки в зависимости от отметки в поле Скидка в обработчике события формы ПриСозданииНаСервере (листинг 4.68).

Листинг 4.68. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Если Объект.Скидка Тогда
        Элементы.СуммаСкидки.Шрифт = Новый Шрифт(,, Истина);
        Элементы.СуммаСкидки.ЦветТекста = WebЦвета.ЗеленаяЛужайка;

    Иначе
        Элементы.СуммаСкидки.Шрифт = Новый Шрифт();
        Элементы.СуммаСкидки.ЦветТекста = WebЦвета.Коралловый;

    КонецЕсли;

КонецПроцедуры
```

Этот пример можно посмотреть в демонстрационной конфигурации «13 (вар. 1) Изменение свойств элементов формы, не требующих обращения к серверу».

Запустим «1С:Предприятие», создадим документ об оказании услуги, выберем услугу, клиента и внесем цену услуги. Первоначально флажок Скидка не установлен, сумма услуги равна цене услуги, а сумма скидки отображается красным цветом текста обычной жирности. Внесем сумму скидки – сумма услуги не изменится. Установим флажок Скидка. После этого поле СуммаСкидки будет отображено жирным шрифтом и зеленым цветом текста, а поле СуммаУслуги автоматически пересчитается по заданному нами алгоритму (рис. 4.77).

При снятии флажка Скидка поле СуммаСкидки обнуляется и отображается красным, а поле СуммаУслуги снова пересчитывается.

Посмотрим теперь на показатели производительности (рис. 4.78).

Заметим, что при первом изменении шрифта элементов формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации. Но мы будем изучать стандартную ситуацию, когда при изменении шрифта элементов формы происходит один вызов сервера.

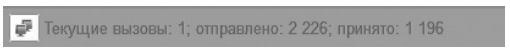


Рис. 4.78. Показатели производительности

Пересчет поля СуммаУслуги происходит на клиенте и не требует обращения к серверу. Также «безболезненно» произойдет и изменение цвета текста поля СуммаСкидки. А вот при изменении свойства Шрифт на клиенте произойдет нежелательный вызов сервера, так как изменение этого свойства влияет на размер элемента формы. Это приведет к пересчету местоположения элементов формы, которое происходит на сервере.

Для наглядности рассмотрим схему программного взаимодействия клиента и сервера (рис. 4.79).

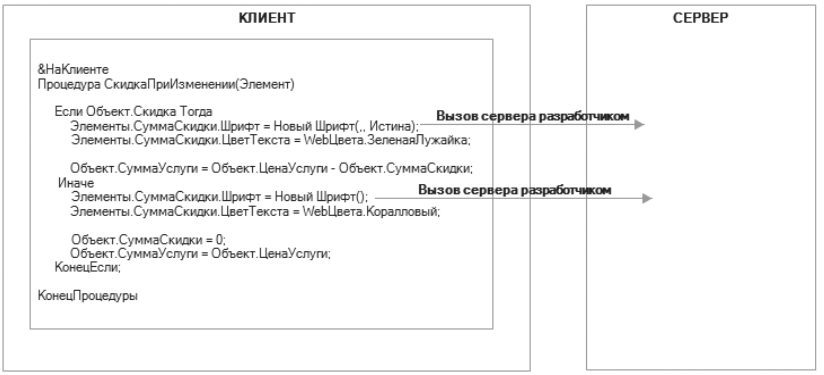


Рис. 4.79. Схема программного взаимодействия сервера и клиента

Второй вариант решения

На самом деле можно обойтись без вызова сервера, если использовать для изменения шрифта поля СуммаСкидки условное оформление формы.

Для этого зададим условное оформление поля СуммаСкидки: жирный шрифт и зеленый цвет текста, в случае если значение поля Скидка истинно (флажок установлен), и красный цвет текста, если значение поля Скидка ложно (рис. 4.80).

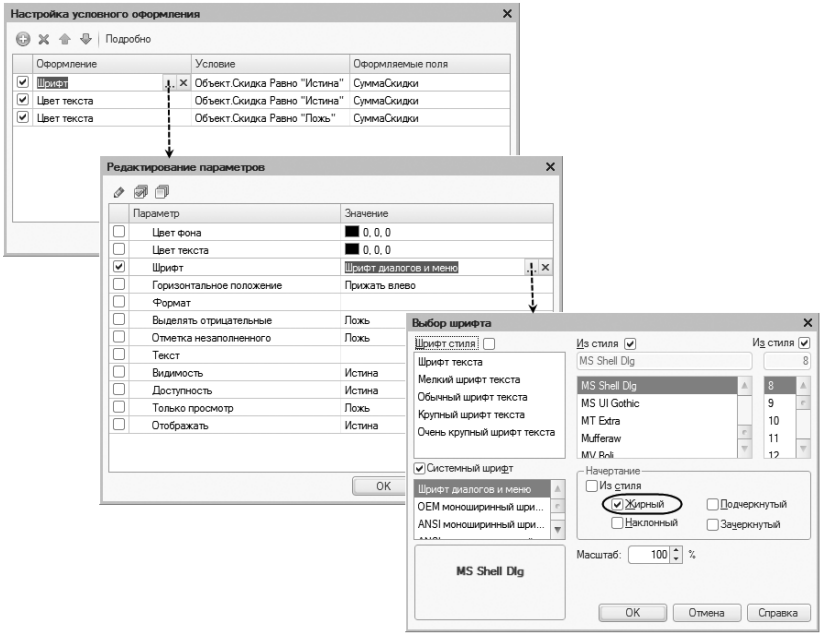


Рис. 4.80. Установка условного оформления для поля «Сумма скидки»

Теперь изменим обработчик события ПриИзменении поля формы Скидка (листинг 4.69).

Листинг 4.69. Процедура «СкидкаПриИзменении()»

```
&НаКлиенте
Процедура СкидкаПриИзменении(Элемент)

    Если Объект.Скидка Тогда
        Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

    Иначе
        Объект.СуммаСкидки = 0;
        Объект.СуммаУслуги = Объект.ЦенаУслуги;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы оставляем только пересчет суммы услуги в соответствии с заданным алгоритмом, а оформление поля мы задали с помощью условного оформления формы (рис. 4.80).

Обработчик события формы ПриСозданииНаСервере можно теперь удалить, так как он больше не нужен.

Этот пример можно посмотреть в демонстрационной конфигурации «13 (вар. 2) Изменение свойств элементов формы, не требующих обращения к серверу».

Запустим «1С:Предприятие», создадим документ об оказании услуги, выберем услугу, клиента и внесем цену услуги. Первоначально флажок Скидка не установлен, сумма услуги равна цене услуги, а сумма скидки отображается красным цветом текста обычной жирности. Внесем сумму скидки – сумма услуги не изменится. Установим флажок Скидка. После этого поле СуммаСкидки будет отображено жирным шрифтом и зеленым цветом текста, а поле СуммаУслуги автоматически пересчитается по заданному нами алгоритму (рис. 4.77).

При снятии флажка Скидка поле СуммаСкидки обнуляется и отображается красным, а поле СуммаУслуги снова пересчитывается.

Заметим, что при первом изменении оформления элементов формы с помощью условного оформления могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации.

Как мы видим, функциональность прикладного решения будет такой же, как и в первом случае, но производительность будет выше, так как изменение шрифта поля формы с помощью условного оформления *не приведет к обращению на сервер.*

Третий вариант решения

В качестве альтернативного варианта при изменении оформления поля СуммаСкидки можно использовать одно из свойств, изменение которых не требует обращения к серверу. Например, свойство ЦветРамки вместо свойства Шрифт.

Попутно сделаем еще одно полезное усовершенствование. Дело в том, что программное изменение оформления поля СуммаСкидки выполняется два раза: при изменении поля Скидка

(в обработчике события ПриИзменении) и при создании формы документа (в обработчике события ПриСозданииНаСервере). Поэтому в первом варианте решения один и тот же код дублировался в каждом из этих обработчиков.

Но лучше «навести порядок» в модуле формы и перенести код, изменяющий оформление поля, в отдельную процедуру. И затем вызывать ее из обработчиков событий ПриИзменении и ПриСозданииНаСервере.

Какой же будет директива компиляции этой процедуры? Ведь она должна вызываться как с сервера, так и с клиента. Если мы предварим процедуру директивой &НаКлиенте, то мы не сможем вызывать ее с сервера, из обработчика события ПриСозданииНаСервере. Точнее, модуль формы не пройдет синтаксическую проверку в контексте сервера.

Если мы напишем директиву &НаСервере, то получим лишний вызов сервера с клиента при возникновении события поля ПриИзменении. Хотя изменение оформительских свойств поля формы вполне может обойтись без вызова сервера.

Единственная подходящая директива – &НаКлиентеНаСервереБезКонтекста. Процедура, описанная такой директивой компиляции, будет выполняться то на сервере, то на клиенте – в зависимости от того, откуда она была вызвана. Но, поскольку процедура будет выполняться без контекста формы, мы должны самостоятельно передать этот контекст в параметре типа УправляемаяФорма.

Итак, добавим в модуль формы процедуру УстановитьОформление() и перенесем в нее код, изменяющий оформление поля СуммаСкидки в зависимости от отметки в поле Скидка (листинг 4.70).

Листинг 4.70. Процедура «УстановитьОформление()»

```
&НаКлиентеНаСервереБезКонтекста
Процедура УстановитьОформление(Форма)

    Если Форма.Объект.Скидка Тогда
        Форма.Элементы.СуммаСкидки.ЦветРамки = WebЦвета.ЗеленаяЛужайка;
    Иначе
        Форма.Элементы.СуммаСкидки.ЦветРамки = WebЦвета.Коралловый;
    КонецЕсли;

КонецПроцедуры
```

В этой процедуре в параметре `Форма` мы получаем контекст формы документа и, используя этот контекст, получаем доступ к коллекции элементов формы (`Форма.Элементы`) и к реквизиту документа Скидка (`Форма.Объект.Скидка`).

Затем изменим обработчик события формы ПриСозданииНаСервере (листинг 4.71).

Листинг 4.71. Обработчик события «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    УстановитьОформление(ЭтотОбъект);

КонецПроцедуры
```

А в обработчике события ПриИзменении поля формы Скидка оставим только пересчет поля СуммаУслуги (листинг 4.72).

Листинг 4.72. Процедура «СкидкаПриИзменении()»

```
&НаКлиенте
Процедура СкидкаПриИзменении(Элемент)

    Если Объект.Скидка Тогда
        Объект.СуммаУслуги = Объект.ЦенаУслуги - Объект.СуммаСкидки;

    Иначе
        Объект.СуммаСкидки = 0;
        Объект.СуммаУслуги = Объект.ЦенаУслуги;

    КонецЕсли;

    УстановитьОформление(ЭтотОбъект);

КонецПроцедуры
```

Запустим «1С:Предприятие», откроем документ об оказании услуги, в котором установлена скидка. Флажок Скидка установлен, и поле СуммаСкидки обрамлено зеленой рамкой. При снятии отметки в поле Скидка рамка поля СуммаСкидки становится красной и значение поля обнуляется. После этого поле СуммаУслуги

Этот пример можно посмотреть в демонстрационной конфигурации «13 (вар. 3) Изменение свойств элементов формы, не требующих обращения к серверу».

пересчитывается. Также оно пересчитывается при любом изменении цены услуги и суммы скидки в соответствии с заданным алгоритмом (рис. 4.81).

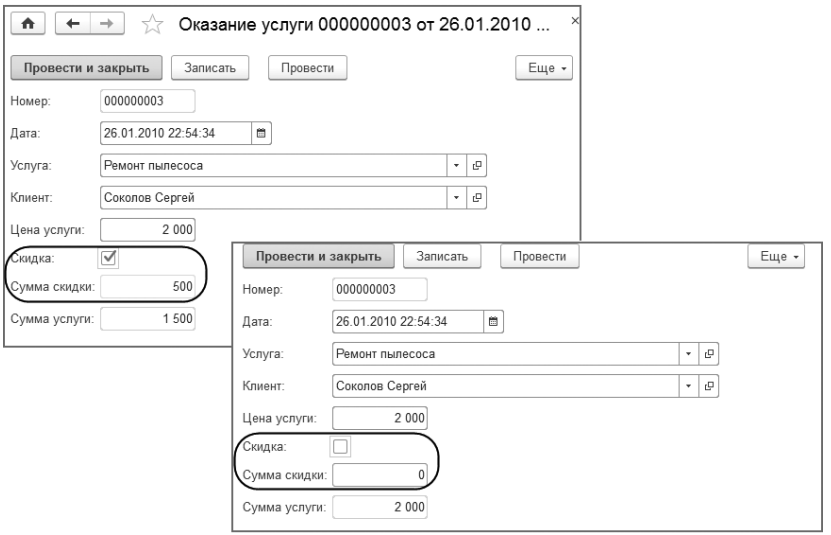


Рис. 4.81. Оформление поля «Сумма скидки» в зависимости от состояния флажка «Скидка»

Как мы видим, функциональность прикладного решения будет практически такой же, как и в первом варианте, но производительность будет выше, так как изменение свойства ЦветРамки на клиенте *не приведет к обращению на сервер.*

Заметим, что при первом изменении цвета рамки элементов формы могут быть лишние серверные вызовы за счет кеширования платформой необходимой информации.

Резюме

Таким образом, если логика интерактивной работы формы требует изменения ее внешнего вида, *лучше использовать те свойства, изменение которых на клиенте не приводит к обращениям на сервер.*

Размер шрифта учитывается при автоматическом определении размеров элементов формы, поэтому при программном изменении шрифта на клиенте выполняется обращение к серверу.

Изменение размера шрифта, выполненное с помощью условного оформления формы и динамического списка, не влияет на размер элемента формы. Поэтому *рекомендуется заменить программное изменение шрифта на клиенте на использование условного оформления.*

Рекомендуется не использовать формы, меняющие состав и положение элементов. И вообще, если есть такая возможность, лучше не использовать свойства и методы встроенного языка, приводящие к вызову сервера (это указано в синтакс-помощнике).

Таким образом, из показанных вариантов решения задачи более эффективным будут второй вариант – за счет изменения шрифта с помощью условного оформления, и третий – за счет изменения свойства поля формы ЦветРамки вместо свойства Шрифт. В результате *в обоих случаях оформление поля на клиенте не потребовало обращения к серверу.*

ЧАСТЬ 5

Мобильный клиент

Оглавление

Глава 5.1. Что такое мобильный клиент

Начиная с версии платформы 8.3.12 фирма «1С» реализовала еще одну технологию для разработки приложений, которые работают на мобильных устройствах. Наряду с уже существовавшей мобильной платформой теперь появился мобильный клиент.

Мобильный клиент не заменяет мобильную платформу. У каждой из этих технологий – свое предназначение и своя «целевая аудитория». Поэтому не стоит их путать.

Мобильная платформа позволяет создавать специализированные мобильные офлайн-приложения, обладающие удобным и функциональным мобильным интерфейсом. Эти приложения разрабатываются для решения конкретных мобильных задач и адаптированы специально под них. Мобильная платформа хорошо подходит для разработки автономных рабочих мест сотрудников, которые находятся вне компании и не имеют надежного интернет-соединения с офисом. Однако такие рабочие места, как правило, обладают ограниченной функциональностью, меньшей, чем функциональность «основного» приложения. К тому же они не обеспечивают онлайн-взаимодействие с информационной базой и требуют периодической синхронизации данных с бэк-офисом.

Мобильный клиент – это аналог тонкого клиента для мобильных устройств, который обладает интерфейсом, аналогичным мобильной платформе. Мобильный клиент позволяет в режиме онлайн получить доступ с мобильных устройств практически ко всей функциональности приложений, для доступа к которым используется тонкий клиент или веб-клиент.

Мобильным клиентом удобно пользоваться там, где обязательно требуется онлайн-доступ в систему с мобильных устройств. При этом данные, введенные на мобильном устройстве, будут попадать непосредственно в «общую» базу данных без дополнительной синхронизации, как если бы их вводили на обычном компьютере, подключившись к информационной базе с помощью тонкого клиента.

Мобильный клиент будет также полезен для пользователей сервисов, работающих на базе технологии 1С:Fresh.

Мобильный клиент взаимодействует с информационной базой по протоколу HTTP. Это означает, что мобильный клиент не поддерживает прямое подключение и может работать только с теми информационными базами, которые опубликованы на веб-сервере. В отличие от приложений, разработанных на мобильной платформе, мобильный клиент работает только в том случае, если между мобильным устройством и веб-сервером существует соединение. Мобильный клиент не может работать в режиме офлайн.

Несомненным плюсом мобильного клиента является поддержка всех возможностей, специфичных для работы на мобильных устройствах (доступ к фотокамере, геопозиционированию, PUSH-уведомлениям и т. д.).

А также мобильный клиент поддерживает технологию расширений конфигурации. Поэтому все характерные для мобильных устройств доработки можно реализовывать в расширении, не затрагивая исходную конфигурацию.

Кроме того, при работе с помощью мобильного клиента не требуется обеспечивать точное соответствие версии мобильного клиента и версии расширения веб-сервера или сервера «1С:Предприятия».

Глава 5.2. Адаптация конфигураций для работы в мобильном клиенте

Общий подход

Как уже говорилось, мобильный клиент обладает интерфейсом, аналогичным мобильной платформе. Поскольку мобильный клиент работает на мобильных устройствах, его интерфейс обладает специфическим набором особенностей и ограничений. Прежде всего эти ограничения касаются размеров экранов и, соответственно, размеров отображаемых на них форм.

Поэтому, хотя мобильный клиент обеспечивает доступ ко всей функциональности конфигураций, его интерфейс отличается от привычного интерфейса в тонком или веб-клиенте. Но при этом мобильный клиент не использует «особенные», специально разработанные для него формы, а отображает те же самые формы, которые используются и «настольными» клиентами. Таким образом, мобильный клиент реализует ту же функциональность, что и «прежние» клиенты, но с другим интерфейсом.

Как же получается этот «новый интерфейс»? Оказывается, платформа львиную долю работы делает сама. Имея то формализованное описание форм, которое имеется в конфигураторе, она компонует формы таким образом, чтобы обеспечить удобную работу с ними на маленьких экранах мобильных устройств.

Но, конечно, платформа не может полностью обеспечить нужный внешний вид форм, поэтому ей нужны «подсказки» от разработчика, чтобы она лучше понимала, как показывать формы на мобильных устройствах. То есть конфигурацию необходимо адаптировать к работе в мобильном клиенте. Эта адаптация потребует значительно меньше усилий, чем разработка приложения на мобильной платформе. Этот момент, несомненно, является большим плюсом для разработчиков.

Все эти доработки можно разделить на два основных направления.

Первое – это избавиться от особенных и специфических интерфейсных решений, больше полагаясь на автоматическую компоновку форм, выполняемую платформой на основе информации о типах данных. Такими специфическими решениями могут быть фиксированные размеры полей, жестко установленная горизонтальная группировка элементов и тому подобное.

Другое направление – это подсказать мобильному клиенту дополнительную информацию об элементах формы. Это потребуется в первую очередь для нестандартных или больших форм, где автоматической адаптации и расстановки элементов будет недостаточно. В этих случаях полезно вручную указать, какие из элементов формы являются более и менее важными, как отображать списки и т. п.

Для того чтобы новые свойства форм и элементов «вступили в силу», режим совместимости конфигурации должен быть установлен не ниже чем с версией 8.3.7.

Также нужно проанализировать те места прикладного решения, в которых алгоритмы работы для тонкого клиента и для веб-клиента различаются. Это нужно сделать для того, чтобы указать, какой из алгоритмов будет использоваться при работе в мобильном клиенте.

О новых свойствах форм и их элементов для адаптации к работе в мобильном клиенте можно прочитать в документации «1С:Предприятие 8.3.12. Руководство разработчика», глава 26 «Разработка для мобильных устройств», разделы 26.3.4.3, 26.3.4.4.8, 26.5.4, а также глава 7 «Формы», раздел 7.7.13.9.

Такие фрагменты кода конфигурации в модулях на клиенте должны быть обрамлены директивой компиляции `МобильныйКлиент`. А функциональность, недоступная на мобильном клиенте, должна быть закрыта директивой компилятора `НеМобильныйКлиент`.

Интерфейс мобильного клиента

При разработке интерфейса мобильного клиента прежде всего надо понимать, что при работе на мобильном устройстве (на планшете и в особенности на смартфоне) мы постоянно сталкиваемся с дефицитом рабочего места. Поэтому размеры элементов форм должны автоматически подстраиваться к размеру мобильного устройства.

По расположению элементов формы должны быть вытянуты в длину и ограничены по ширине, так как вертикальная прокрутка форм вполне ожидаема и привычна, в то время как горизонтальная прокрутка на мобильных устройствах не поддерживается.

Как уже говорилось, чтобы минимизировать усилия разработчика, в платформе реализован ряд механизмов для самостоятельной адаптации форм к работе на маленьких экранах мобильных устройств.

Для демонстрации этих возможностей возьмем конфигурацию ДемонстрационнаяКонфигурацияУправляемоеПриложение, для которой уже собран мобильный клиент. Работу мобильного клиента мы будем тестировать на планшете Samsung Galaxy Tab2 (10.1) GT-P5110, работающем под управлением операционной системы Android.

Свойства форм и их элементов (которые мы подробно рассмотрим ниже), отвечающие за адаптацию к работе в мобильном клиенте, пока установлены в значение Авто. Итак, посмотрим, как изменение этих свойств повлияет на отображение форм на мобильных устройствах.

Сначала опубликуем нашу конфигурацию на веб-сервере. Для этого нужно выполнить команду конфигуратора Администрирование – Публикация на веб-сервере... (рис. 5.1).

В появившемся диалоге в поле Имя нужно задать имя виртуального каталога на веб-сервере, в который будет выполнена публикация.

Поле Веб-сервер автоматически заполнилось единственным установленным на нашем компьютере веб-сервером Apache 2.2.

В поле Каталог нужно указать физический каталог компьютера, в котором будет находиться файл публикации мобильного приложения.

Затем нужно нажать кнопку Опубликовать.

После этого запустим на планшете apk-файл для установки мобильного клиента. Приложение с пиктограммой установится и добавится в список приложений планшета.

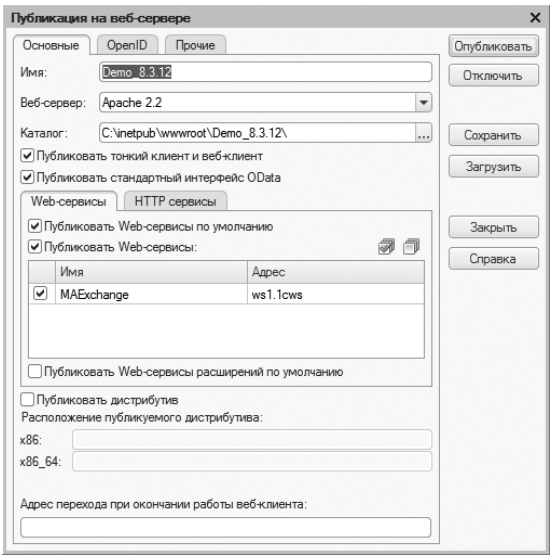


Рис. 5.1. Публикация конфигурации на веб-сервере

Откроем его и добавим новое мобильное приложение, нажав на кнопку со значком «+» в правом верхнем углу экрана в строке Приложения. В появившемся окне зададим произвольное наименование приложения. В поле Веб-сервер необходимо указать URL веб-сервера, на котором опубликована конфигурация, и нажать кнопку Готово в правом верхнем углу экрана (рис. 5.2).

В приведенном примере на рис. 5.2 в поле Веб-сервер указано: http://<IP-адрес компьютера в беспроводной сети>/<Каталог веб-сервера, на котором опубликовано мобильное приложение>.

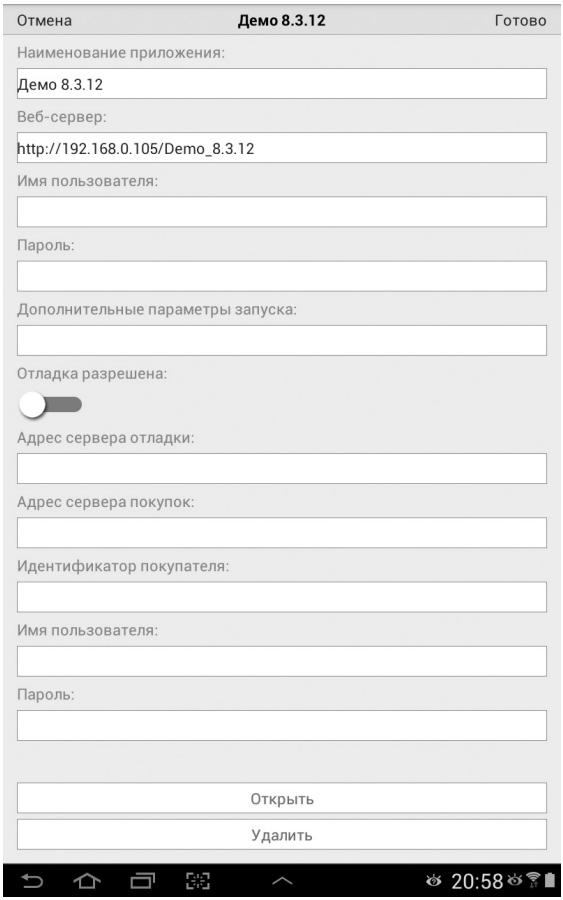


Рис. 5.2. Добавление тестовой конфигурации для мобильного клиента

Поведение таблиц при сжатии по горизонтали

Сначала, ничего не меняя, посмотрим, как выглядят формы списков в случае полной автоматической адаптации к размеру нашего планшета.

Запустим наше приложение (Демо 8.3.12) от имени пользователя с именем Администратор. Жестом пролистывания слева откроем список разделов приложения.

В разделе Продажи по команде Продажи откроем список продаж товаров (список документов РасходТовара). Оценим вид списка продаж в вертикальной и горизонтальной ориентации планшета (рис. 5.3).

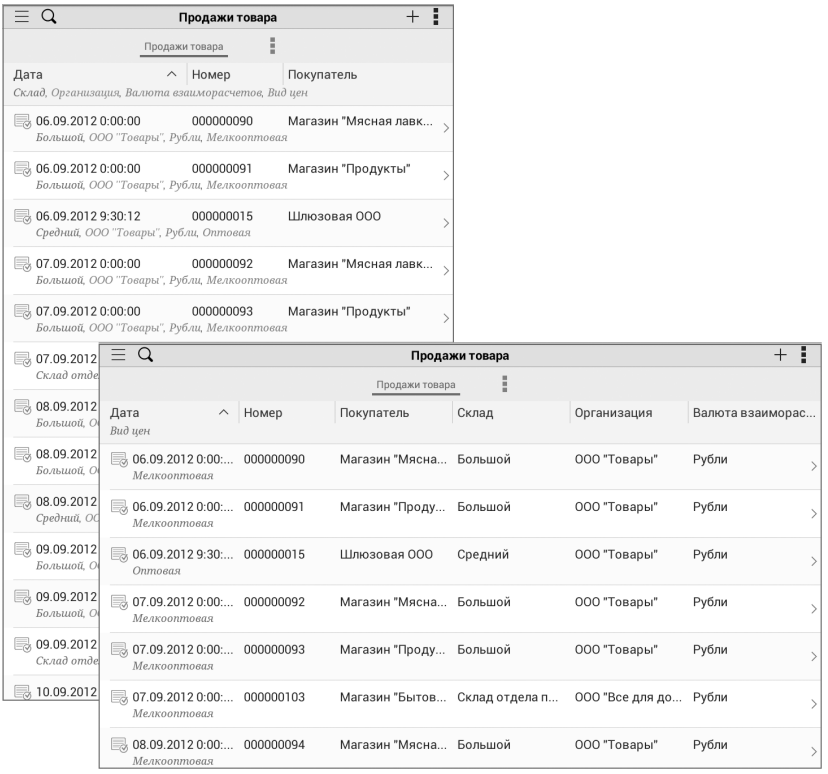


Рис. 5.3. Список продаж товаров

Мы видим, что вид списка продаж (как и любых других списков) изменился по сравнению с экраном стационарного компьютера. Таблицы списка имеют только вертикальную прокрутку и сжимаются по горизонтали. При этом колонки, не умещающиеся в одну строку, не отображаются, а их значения переносятся на следующую строку и отображаются мелким курсивом разного цвета через запятую.

Рассмотрим подробно, какие свойства на мобильном клиенте определяют такое поведение.

Ключевым свойством, помогающим формам (самостоятельно или с помощью разработчика) адаптироваться к экранам мобильных устройств, является свойство ВажностьПриОтображении. Это свойство есть у полей формы, групп, страниц, таблиц и других элементов формы. Оно принимает значения перечисления ВажностьПриОтображении:

- Авто. Элементам формы, связанным с основным реквизитом формы, устанавливается важность ОченьВысокая. Важность Высокая присваивается командным панелям, источником которых выступает сама форма, и т. д.

  Подробнее об этом можно прочитать в документации «1С:Предприятие 8.3.12. Руководство разработчика», глава 26 «Разработка для мобильных устройств», раздел 26.3.4.3, а также глава 7 «Формы», раздел 7.7.13.9.
- Высокая. Важность высокая.
- Низкая. Важность низкая.
- Обычная. Важность обычная.
- ОченьВысокая. Важность очень высокая.
- ОченьНизкая. Важность очень низкая.

При адаптации списков к ширине экранов мобильных устройств поведение таблиц формы, отображающих данные списков, определяется свойством ПоведениеПриСжатииПоГоризонтали. Оно принимает значения перечисления ПоведениеТаблицыПриСжатииПоГоризонтали:

- Авто:
  - в мобильной платформе используется значение СкрыватьЭлементыПоВажности;
  - в мобильном клиенте – ПереноситьЭлементыПоВажности.
- СкрыватьЭлементыПоВажности. При адаптации к ширине экрана колонки с наименьшей важностью скрываются. В пределах одного уровня важности скрываются элементы, последовательно расположенные справа налево.

- ПереноситьЭлементыПоВажности. При адаптации к ширине экрана колонки с наименьшей важностью скрываются, а их значения отображаются в отдельной строке мелким шрифтом с курсивным начертанием с использованием нескольких цветов. В пределах одного уровня важности переносятся элементы, расположенные справа налево. Элементы в строке располагаются в порядке убывания важности.

Таким образом, хотя у таблицы формы списка документов РасходТовара свойство ПоведениеПриСжатииПоГоризонтали стандартно установлено в значение Авто, на мобильном клиенте оно трактуется как ПереноситьЭлементыПоВажности. Поэтому при адаптации к ширине экрана нашего планшета мы видим поведение, показанное на рис. 5.3.

Стандартно у всех колонок таблицы списка свойство ВажностьПриОтображении принимает значение Авто. На мобильном клиенте это трактуется как ОченьВысокая важность, так как колонки отображают данные основного реквизита формы – динамического списка. Поскольку важность у всех колонок пока одинаковая, то переносятся не поместившиеся по ширине элементы, начиная справа, – налево. А порядок их следования друг за другом определяется порядком колонок в таблице списка, заданным при конфигурировании формы.

Теперь откроем в редакторе форму списка документа РасходТовара, попробуем изменить важность колонок и посмотрим, как изменится вид списка в мобильном клиенте.

Для помощи разработчику в командной панели редактора формы есть блок кнопок, позволяющих оценить в режиме предварительного просмотра, как будет выглядеть форма на конкретном мобильном устройстве при вертикальной или горизонтальной ориентации экрана.

Для этого сначала с помощью кнопки  нужно выбрать платформу, на которой выполняется прикладное решение, – в нашем случае это мобильный клиент. Затем, нажав на кнопку  , можно выбрать конкретное мобильное устройство (поскольку нашего планшета нет в списке устройств, мы выбрали похожее по размеру устройство). И затем с помощью кнопки  можно поменять ориентацию экрана (рис. 5.4).

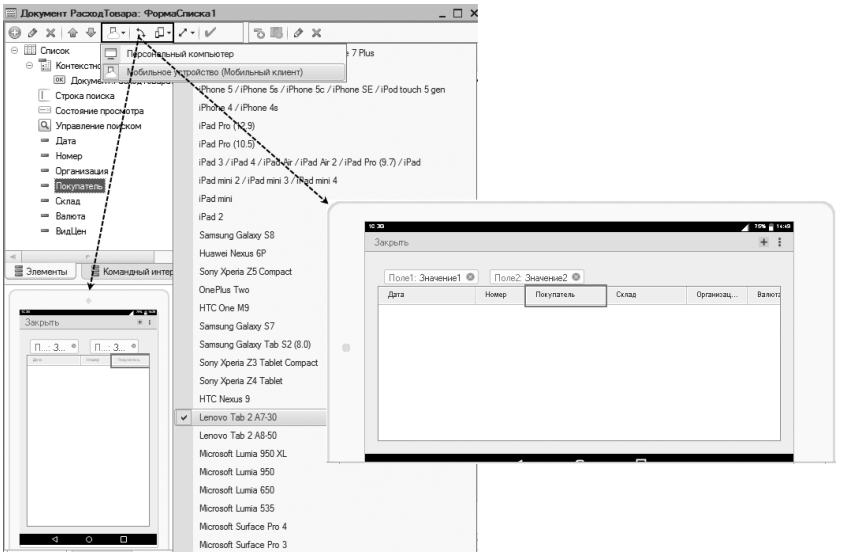


Рис. 5.4. Предварительный просмотр формы на мобильном устройстве в редакторе формы

Установим свойство ВажностьПриОтображении у колонок таблицы формы Список в следующие значения:

- Дата – Авто,
- Номер – Авто,
- Организация – ОченьНизкая,
- Покупатель – Авто,
- Склад – ОченьНизкая,
- Валюта – Низкая,
- ВидЦен – Обычная.

Обновим конфигурацию базы данных (F7) и снова опубликуем конфигурацию на веб-сервере. После этого запустим наше демонстрационное приложение на планшете и откроем список продаж товаров. Мы видим, что вид списка изменился.

Поскольку колонкам динамического списка со значением Авто присваивается ОченьВысокая важность, то по мере убывания важности колонки будут следовать друг за другом в следующем

порядке: Дата, Номер, Покупатель, ВидЦен, Валюта, Склад, Организация. При этом последние не поместившиеся по ширине колонки будут переноситься на следующую строку (рис. 5.5).

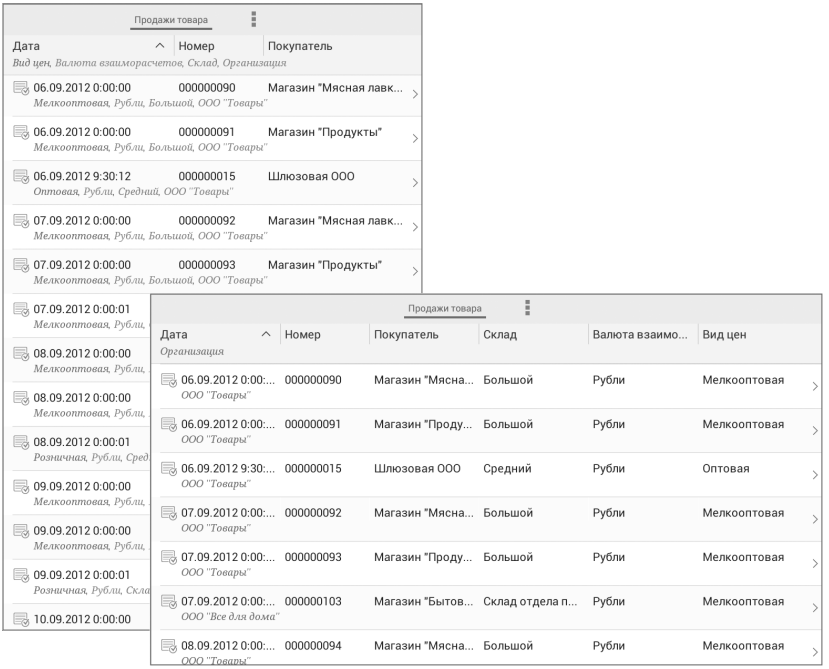


Рис. 5.5. Список продаж товаров

Таким образом, если мы хотим изменять порядок следования колонок в списке, то должны соответствующим образом установить свойство ВажностьПриОтображении у колонок таблицы формы, отображающей данные списка. А свойство ПоведениеПриСжатииПоГоризонтали можно не изменять, так как оно стандартно установлено в значение Авто, что на мобильном клиенте трактуется как ПереноситьЭлементыПоВажности.

Сворачивание элементов форм по важности

Теперь посмотрим, как адаптируются к размерам мобильных устройств формы документов, элементов справочников и т. п.

Прежде всего форма пытается подстроиться к ширине экрана на мобильном устройстве. Для тех элементов формы, ширина которых жестко задана в конфигурации и превышает фактическую ширину экрана мобильного устройства, ширина уменьшается принудительно – таким образом, чтобы элемент поместился (по ширине) в текущей ориентации мобильного устройства без горизонтальной прокрутки.

Поэтому при разработке форм фиксированную ширину элементов лучше не использовать (свойство Ширина должно быть равно нулю, а свойство АвтоМаксимальнаяШирина – установлено).

А также для групп формы, элементы которых на стационарном компьютере требуется отображать горизонтально, свойство Группировка должно быть установлено в значение Горизонтальная если возможно. Потому что если свойство Группировка установлено в значение Горизонтальная всегда, то на узких экранах мобильных устройств элементы таких групп могут быть не видны, так как горизонтальная прокрутка форм на мобильном устройстве не используется.

Например, в редакторе формы документа РасходТовара мы видим, что реквизиты документа помещены в две группы с вертикальной группировкой: ЛеваяКолонка и ПраваяКолонка. В свою очередь, эти две группы объединены родительской группой Шапка. Тип группировки у этой группы не определен, что трактуется как значение по умолчанию – Горизонтальная если возможно (см. рис. 5.6).

В окне предварительного просмотра мы видим, что на экране стационарного компьютера реквизиты документа (при достаточной ширине и разрешении экрана) будут выводиться горизонтально, в две вертикальные колонки (рис. 5.6).

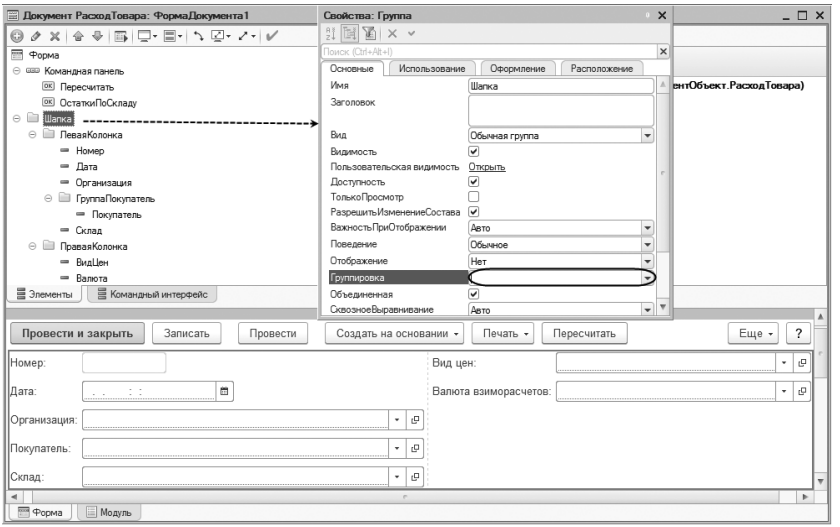


Рис. 5.6. Форма документа «Расход товара» в редакторе формы

Но поскольку мобильные устройства значительно ограничены по ширине, то на них левая и правая колонка реквизитов будут располагаться вертикально друг под другом. То есть, поскольку горизонтальная группировка элементов невозможна, она превращается в вертикальную (рис. 5.7).

Но мы видим, что такое отображение также может быть неудобно, особенно в случае документа с большой табличной частью. Попробуем оптимизировать форму документа, чтобы не приходилось ее все время прокручивать по вертикали. Рассмотрим, с помощью каких свойств это можно сделать.

В процессе адаптации к размерам мобильных устройств платформа пытается уменьшить количество отображаемых элементов формы, чтобы улучшить ее внешний вид, сделать форму более лаконичной и акцентировать внимание пользователя на самой важной информации. Для управления возможностью такой перестройки предназначено свойство формы СворачиваниеЭлементовПоВажности.

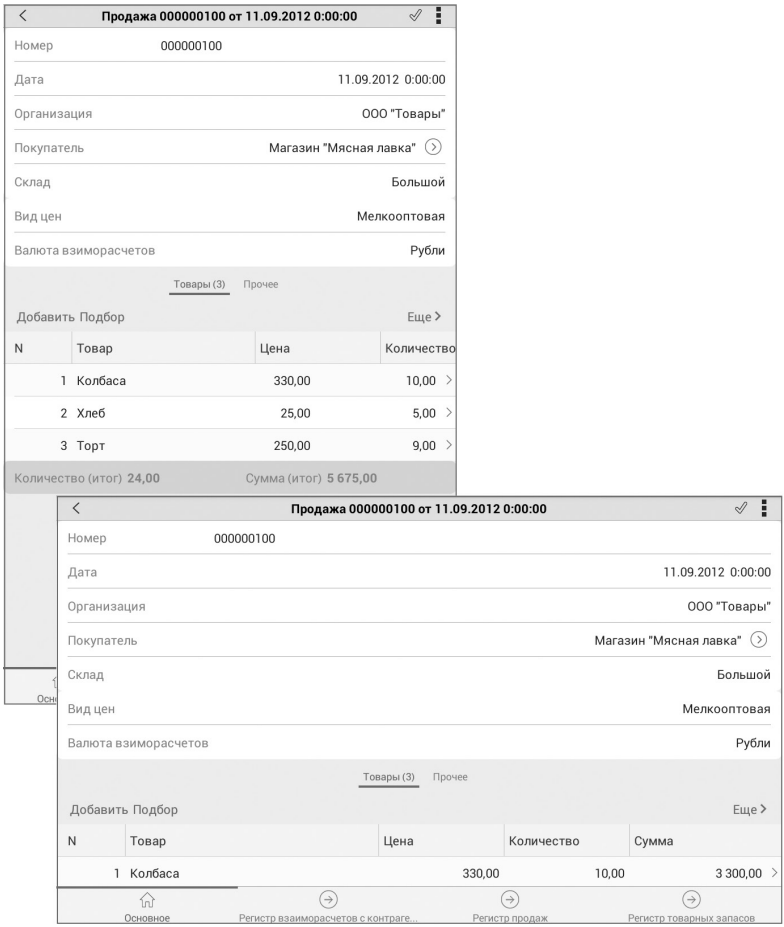


Рис. 5.7. Форма документа «Расход товара»

Свойство принимает значения перечисления СворачиваниеЭлементовФормыПоВажности:

- Авто. Интерпретируется как Использовать.
- Использовать. Сворачивание элементов формы по важности используется.
- НеИспользовать. Сворачивание элементов формы по важности не используется.

Для определения того, какие элементы формы отображать полностью, а какие сворачивать, используется свойство ВажностьПриОтображении, о котором было рассказано выше. Изменяя значение этого свойства у элементов формы, разработчик может добиться нужного вида формы на экране устройства. Значение этого свойства мобильный клиент обрабатывает исходя из принципа, что более важным элементам отводится больше места в форме:

- Если более важный элемент расположен под менее важными и эти менее важные элементы занимают более трех строк формы, то менее важные элементы объединяются и помещаются в сворачиваемую группу.
- Если более важный элемент расположен внутри иерархии групп или страниц и при этом в разных местах иерархии над ним расположены менее важные элементы, занимающие более трех строк, в каждом уровне иерархии формируется сворачиваемая группа.
- Если имеются несколько элементов более высокой важности, то менее важные элементы, находящиеся между ними, помещаются в сворачиваемую группу, если менее важные элементы занимают более трех строк.
- И т. д. Подробнее об этом можно прочитать в документации «1С:Предприятие 8.3.12. Руководство разработчика», глава 26 «Разработка для мобильных устройств», раздел 26.3.4.3, а также глава 7 «Формы», раздел 7.7.13.9.

Продемонстрируем вышесказанное на примере.

Установим свойство ВажностьПриОтображении у поля Организация формы документа РасходТовара в значение Низкая. У всех остальных элементов формы ВажностьПриОтображении стандартно установлена в Авто. У самой формы свойство СворачиваниеЭлементовПоВажности также принимает значение Авто. Это значит, что сворачивание элементов формы по важности используется.

Обновим конфигурацию базы данных (F7) и опубликуем конфигурацию на веб-сервере. После этого запустим наше демонстрационное приложение на планшете и откроем форму документа Продажа (РасходТовара).

Мы видим, что вся группа реквизитов, относящихся к группе ЛеваяКолонка (Номер, Дата, Организация, Покупатель, Склад,

см. рис. 5.6) показана в свернутом виде. Заголовок этой группы формируется перечислением через запятую заголовков всех элементов, входящих в свертываемую группу. Мы можем раскрыть ее на отдельном экране планшета, нажав на заголовок группы или на специальный значок справа от заголовка (рис. 5.8).

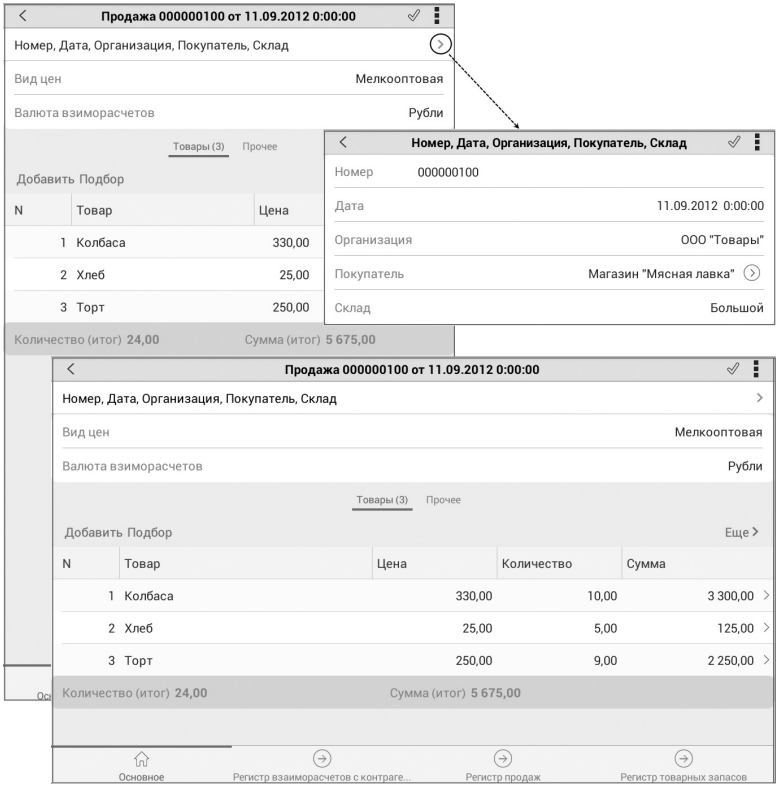


Рис. 5.8. Форма документа «Расход товара»

Так произошло потому, что группе формы ЛеваяКолонка стандартно присвоена важность Авто при отображении. В этом случае группа имеет самую высокую важность из всех входящих в нее элементов, то есть в результате важность у группы Низкая, как и у поля формы Организация, входящего в группу. Таким образом, и у всех остальных элементов группы, имеющих важность Авто , она становится Низкая.

А поскольку поля группы занимают более трех строк формы, вся группа сворачивается.

Теперь вернем свойство ВажностьПриОтображении у поля формы Организация в значение Авто. А у группы формы ПраваяКолонка свойство ВажностьПриОтображении установим в значение Низкая. В результате вся группа целиком будет отображена в свернутом виде с заголовком в виде перечня входящих в нее элементов: «Вид цен, Валюта взаиморасчетов» (рис. 5.9).

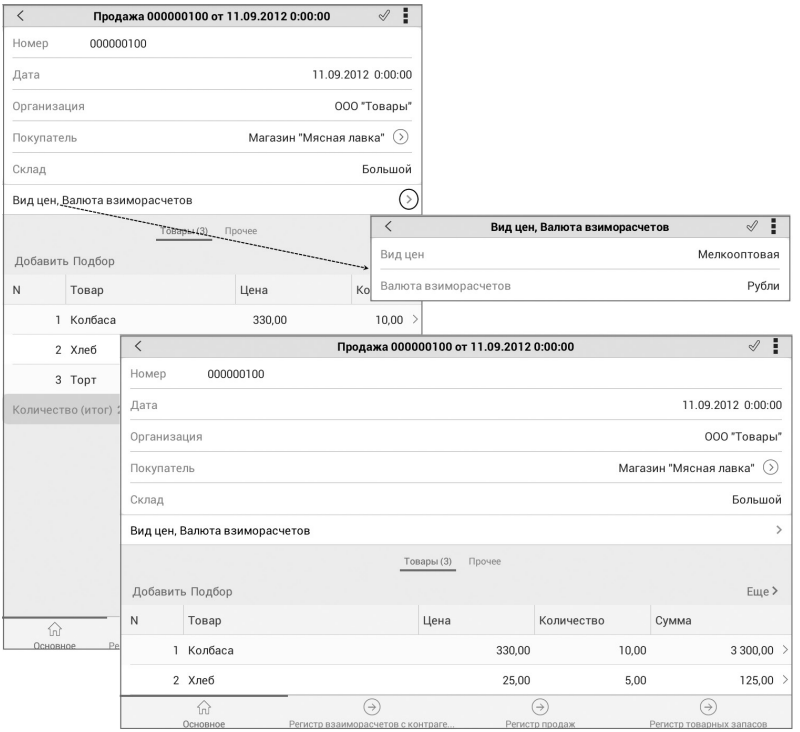


Рис. 5.9. Форма документа «Расход товара»

Использование текущей строки таблицы

Нажатие на строку таблицы в мобильном клиенте имеет свои особенности, которые нужно знать и учитывать. Поведение текущей строки таблицы при этом определяется свойством Исполь-

зованиеТекущейСтроки. Оно принимает значения перечисления ИспользованиеТекущейСтрокиТаблицы:

- Авто. Данное значение трактуется как значение Выбор и на мобильной платформе, и в мобильном клиенте.
- Выбор. В этом случае свойства ТекущаяСтрока, ТекущийЭлемент, ТекущиеДанные определены только во время выполнения:
  - обработчика контекстной команды;
  - событий активизации строки или ячейки;
  - событий редактирования строки.
- В остальное время эти свойства таблицы имеют значение Неопределено. Для команды формы, которой необходимы данные текущей строки, следует явно указать такую необходимость в свойстве команды ИспользованиеТекущейСтроки. Дополнительно необходимо указать таблицу, данные из которой будет использовать команда, с помощью свойства ИспользуемаяТаблица.
- В списке текущая строка визуально определяется кратковременно, во время нажатия на строку.
- При нажатии на строку вызываются события ПриАктивизацииПоля, ПриАктивизацииСтроки, ПриАктивизацииЯчейки, Выбор.
- Данное значение рекомендуется использовать для таблиц, логически не связанных с какими-то другими данными (в том числе и табличными).
- ОтображениеВыделения. В этом случае текущая строка существует всегда, если таблица содержит хотя бы одну строку.
- В списке текущая строка визуально определяется всегда.
- При нажатии на строку вызывается событие ПриАктивизацииПоля. Событие Выбор не вызывается.
- Данное значение рекомендуется использовать для таблиц, которые логически связаны с какими-то другими данными (в том числе и табличными).
- ОтображениеВыделенияИВыбор. В этом случае текущая строка существует всегда, если таблица содержит хотя бы одну строку.
- В списке текущая строка визуально определяется всегда. Также в правой части строки имеется кнопка, при нажатии которой вызываются события ПриАктивизацииПоля и Выбор.

Чтобы продемонстрировать вышесказанное на примере, рассмотрим ситуацию, когда в форме размещены список групп и собственно содержимое иерархического справочника, а при нажатии на группу в первой таблице нужно отобразить во второй таблице список элементов справочника, относящихся к выбранной группе.

Как мы уже знаем, такую автоматическую синхронизацию данных в форме можно поддерживать с помощью свойства СписокГрупп, когда одна таблица (связанная с основным реквизитом формы типа ДинамическийСписок) отображает данные справочника, а другая таблица отображает динамический список, который показывает только иерархию групп отображаемого справочника.

Для примера возьмем форму списка иерархического справочника Контрагенты. В форме списка присутствует таблица Список, связанная с основным реквизитом Список (типа ДинамическийСписок), получающим данные из справочника Контрагенты. Создадим еще один такой же динамический список – реквизит формы ГруппыКонтрагентов. Таблицу, связанную с этим реквизитом в форме, поместим над таблицей Список и укажем эту таблицу в свойстве формы СписокГрупп (рис. 5.10).

Таким образом, в верхней таблице будет отображаться список групп контрагентов, а под ним, в нижней таблице – содержимое текущей группы, выделенной в первой таблице.

В соответствии с требуемым поведением установим свойство ИспользованиеТекущейСтроки таблицы ГруппыКонтрагентов в значение ОтображениеВыделения, так как нам все время нужно видеть в этой таблице выделенную группу контрагентов, а выбор конкретного контрагента производить в нижней таблице.

Обновим конфигурацию базы данных (F7) и опубликуем конфигурацию на веб-сервере. После этого запустим наше демонстрационное приложение на планшете и откроем форму списка контрагентов.

Поведение двух связанных списков соответствует желаемому. При этом у нижней таблицы формы Список свойство ИспользованиеТекущейСтроки мы не меняли, так как стандартно оно установлено в значение Авто, что соответствует значению Выбор (рис. 5.11).

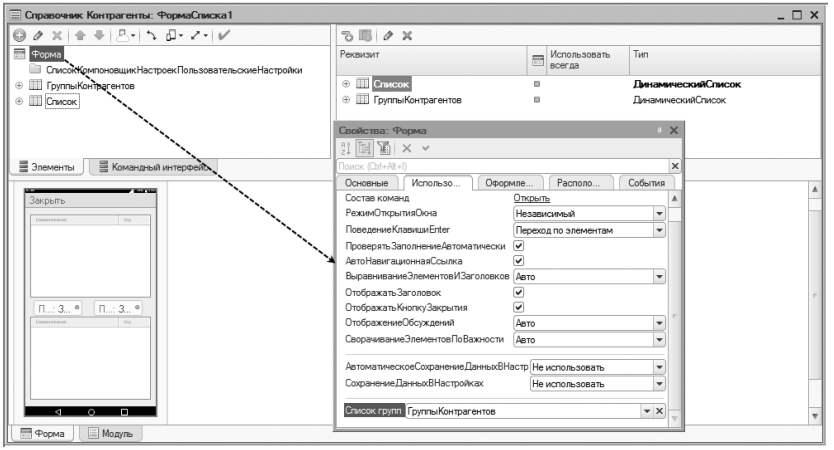


Рис. 5.10. Форма списка справочника «Контрагенты» в конфигураторе

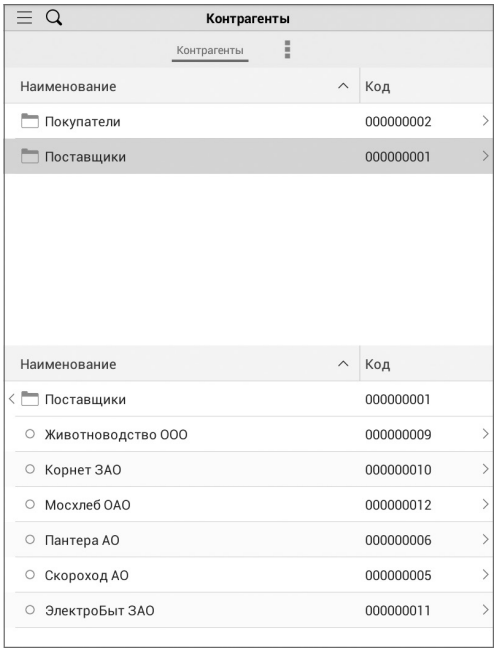


Рис. 5.11. Форма списка контрагентов

Использование текущей строки командой формы

Типичный пример использования командой формы текущей строки таблицы – это, например, когда из формы списка справочника вызывается команда, связанная с текущим элементом этого списка.

Предположим, нам нужно открыть список заказов, сделанных конкретным контрагентом. Для отображения списка контрагентов в форме списка справочника используется таблица Список, связанная с основным реквизитом Список (типа ДинамическийСписок), получающим данные из справочника Контрагенты.

Добавим команду Заказы и перетащим ее в командную панель формы. Установим свойство команды ИспользованиеТекущейСтроки в значение Использует, а в свойстве ИспользуемаяТаблица выберем таблицу Список (рис. 5.12).

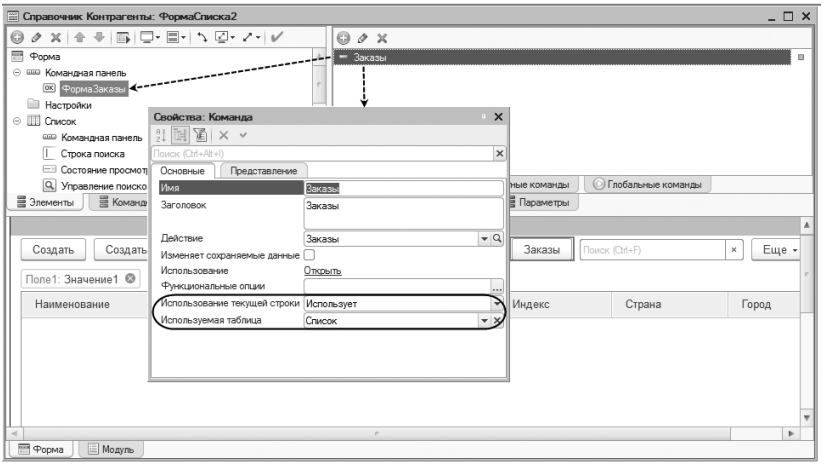


Рис. 5.12. Форма списка справочника «Контрагенты» в конфигураторе

Таким образом мы задали, что команда Заказы будет использовать данные текущей строки таблицы Список. Создадим обработчик команды и заполним его следующим образом (листинг 5.1).

Листинг 5.1. Обработчик команды «Заказы»

```
&НаКлиенте
Процедура Заказы(Команда)

    ПараметрыФормы = Новый Структура("Отбор", Новый Структура("Покупатель",
                        Элементы.Список.ТекущаяСтрока));
    ОткрытьФорму("Документ.Заказ.ФормаСписка", ПараметрыФормы);

КонецПроцедуры
```

В этом обработчике, используя свойство ТекущаяСтрока таблицы Список, мы получаем ссылку на текущий элемент списка контрагентов. И устанавливаем эту ссылку, как значение отбора по полю Покупатель, в структуре параметров формы. Затем открываем форму списка документа Заказ с отбором по данному контрагенту.

Обновим конфигурацию базы данных (F7) и опубликуем конфигурацию на веб-сервере. После этого запустим наше демонстрационное приложение на планшете и откроем форму списка контрагентов.

Стандартно у таблицы списка свойство ИспользованиеТекущейСтроки установлено в значение Авто, поэтому она работает в режиме выбора конкретного контрагента и показывает выделенную строку лишь кратковременно. Но поскольку команда Заказы использует текущую строку таблицы, то она будет доступна из контекстного меню, вызываемого жестом пролистывания с правой стороны экрана нужной строки списка контрагентов.

Вызовем контекстное меню у какого-либо контрагента. Поскольку меню большое, то, чтобы увидеть нашу команду, нажмем Еще и в появившемся списке команд выберем команду Заказы. В результате в новом окне планшета откроется список заказов, сделанных выбранным контрагентом (рис. 5.13).

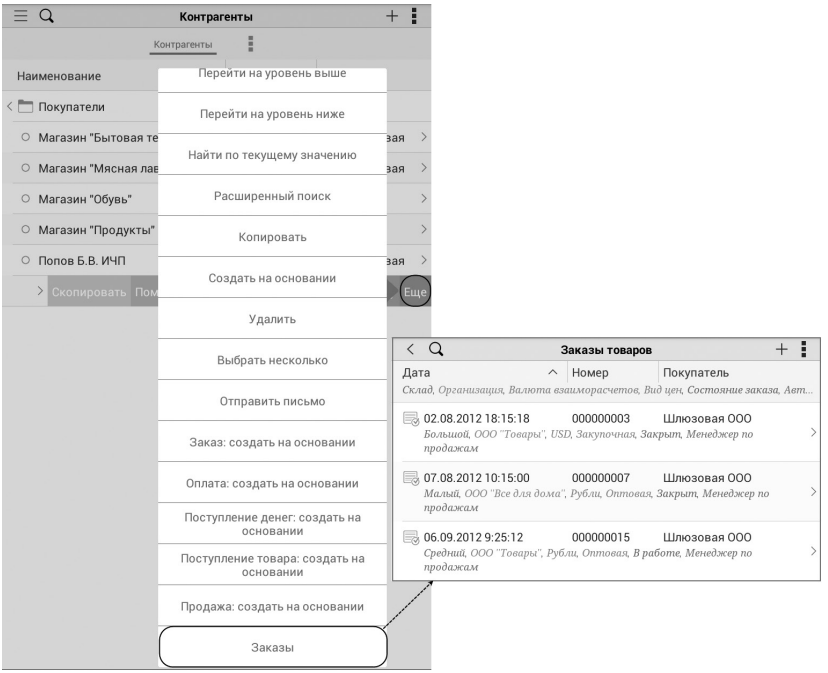


Рис. 5.13. Список заказов контрагента

Использование текущей строки группой формы

Если в форме есть данные, связанные с текущей строкой таблицы, но эти данные носят второстепенный, справочный характер, то такие данные лучше объединять в группы и указывать для групп свойства ИспользованиеТекущейСтроки и ИспользуемаяТаблица по аналогии с командами формы (см. рис. 5.14).

Для каждой используемой таблицы должна быть своя группа данных, так как одна группа может отображать данные только одной таблицы.

В мобильном клиенте, в соответствии с концепцией лаконичности интерфейса, группы, использующие текущую строку таблицы, отображаться не будут. Чтобы их увидеть, нужно вызвать контекстное меню какой-либо строки таблицы и выполнить команду Связанные данные. В результате откроется отдельное окно, в котором будут показаны все группы с данными, которые используют выбранную строку таблицы.

Продемонстрируем вышесказанное на примере.

Предположим, что для экономии места в форме документа Поступление товара мы не хотим постоянно видеть некоторые реквизиты табличной части документа. А хотим отображать их опционально, по специальной команде контекстного меню, вызванного у конкретной строки табличной части.

Для решения этой задачи в редакторе формы документа ПриходТовара перенесем колонки ТоварыАртикул и ТоварыЦена таблицы Товары, отображающей данные табличной части документа, в отдельную группу Дополнительно, которая будет использовать текущую строку этой таблицы (рис. 5.14).

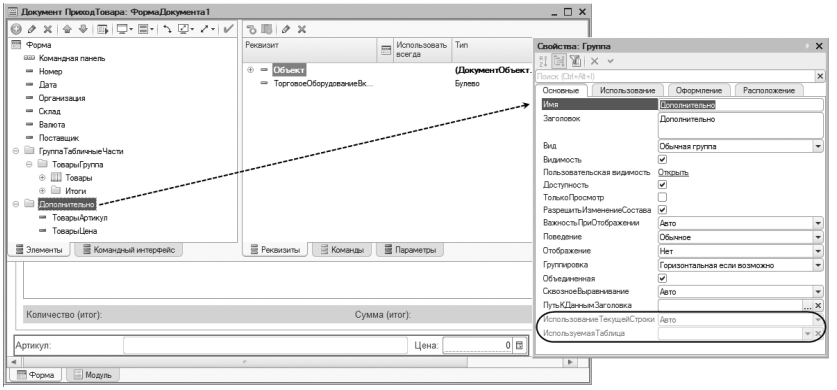


Рис. 5.14. Форма документа «ПриходТовара» в конфигураторе

В результате на планшете мы не увидим в табличной части документа Поступление товара колонок Артикул и Цена. Чтобы увидеть артикул и цену товара в конкретной строке таблицы, выполним команду Связанные данные из контекстного меню этой строки (рис. 5.15).

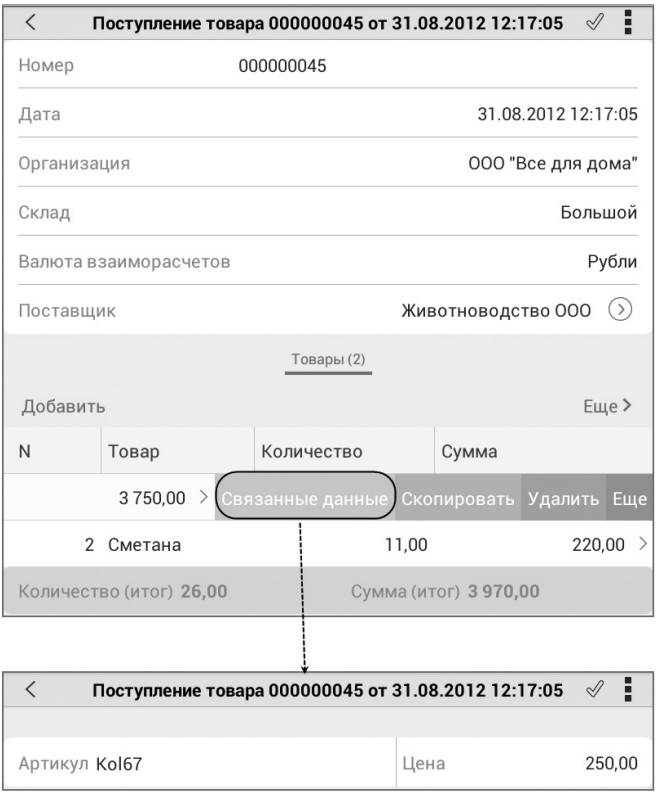


Рис. 5.15. Форма документа «Поступление товара»

Оглавление

Интернет-конференция для начинающих разработчиков
http://devtrainingforum.v8.1c.ru/forum.

Фирма «1С»
123056, Москва, а/я 64, Селезневская ул., 21.
Тел.: (495) 737-92-57, факс: (495) 681-44-07.
1c@1c.ru, http://www.1c.ru/

Издательство ООО «1С-Паблишинг»
127434, Москва, Дмитровское ш., 9.
Тел.: (495) 681-02-21, факс: (495) 681-44-07.
publishing@1c.ru, http://books.1c.ru

Об опечатках просьба сообщать по адресу books.v8@1c.ru.